C++

PROGRAMMING FUNDAMENTALS

by

Tin Mai Zaw

2025 EDITION

Northern City, Yangon, Myanmar

©2025 Northern City. All Rights Reserved

အမှာစာ

ဤစာအုပ်သည် C++ programming ကို အခြေခံမှ စပြီးလေ့လာချင်သူများ ၊ programming foundation ကို ခိုင်ခိုင်မာမာ တည်ဆောက်ချင်သူများအတွက် စာရေးသူက တစ်နှစ်လောက် အချိန်ယူပြီး အသေးစိတ် ရေးသားဖော်ပြထားတဲ့ စာတွေ့လက်တွေ့ C++ programming fundamentals စာအုပ်ဖြစ်ပါတယ်။ ပြည်တွင်းပြည်ပမှာ နေထိုင်လျက်ရှိသော programming ကို စိတ်ဝင်စားသူ အိုင်တီ ညီအကို မောင်နှမ အားလုံးအတွက် ဖြစ်ပါတယ်။ မြန်မာနိုင်ငံ အနယ်နယ်အရပ်ရပ်မှာ ပညာသင်ကြားနေတဲ့ ကွန်ပျူတာ ကျောင်းသူ ကျောင်းသားများ၊ အင်ဂျင်နီယာ အိုင်တီ ကျောင်းသူ ကျောင်းသား အားလုံးကို တစ်ဖက်တလမ်း ကူညီပံ့ပိုးနိုင်အောင် အချိန်ယူ ရေးသားထားတဲ့ စာအုပ် ဖြစ်ပါတယ်။ မကြာခင်မှာ ဒီ စာအုပ်နဲ့အတူ tutorial video များ ၊ github source code တွေ ၊ အသေးစိတ် ရှင်းလင်းချက်တွေနဲ့ ထုတ်ဝေသွားဖို့ စီစဉ်နေပါတယ်။

Programming ကိုတော့လေ့လာချင်တယ်။ ဒါပေမယ့် စာဖတ်တာ ဝါသနာမပါဘူး၊ စာဖတ်အားနည်းတယ်။ Video Tutorial တွေကို ကြည့်ရတာ ပိုကြိုက်တယ် ဆိုရင်တော့ ကျနော်တို့ Northen City Youtube Channel မှာ သွားပြီး C++, Java Programming, Python Programming, HTML,CSS, JavaScript, Laravel စတဲ့ အိုင်တီ ဘာသာရပ် တွေ ကို အခမဲ့ လေ့လာနိုင်ပါတယ်။ Northern City Youtube Channel Link နဲ့ Facebook Page Link တွေကတော့ အောက်ပါအတိုင်းဖြစ်ပါတယ်။

Youtube Channel:

https://www.youtube.com/@northerncity2022

Facebook Page:

https://www.facebook.com/northerncity.programmingspace

မာတိကာ

1.	C++ သမိုင်းအကျဉ်း	စာမှက်နှာ	၅
2.	ဘာလို့ C++ ကို လေ့လာသင့်သလဲ?	စာမှက်နှာ	၈
3.	Hardware & Software Requirements	စာမှက်နှာ	၁၁
4.	Data Types	စာမှက်နှာ	၁၈
5.	Variables	စာမှက်နှာ	၂၃
6.	Basic Calculations	စာမှက်နှာ	၃၂
7.	Conditional Statements	စာမှက်နှာ	၃၇
8.	Loops	စာမှက်နှာ	၄၂
9.	Arrays	စာမှက်နှာ	၄၈
10.	Functions	စာမှက်နှာ	၅၄
11.	Random	စာမှက်နှာ	၆၂
12.	Date	စာမှက်နှာ	၇၆
13.	String Manipulations	စာမှက်နှာ	၈၃
14.	Class	စာမှက်နှာ	၉၅
15.	Methods	စာမှက်နှာ	၁၀၄
16.	OOP	စာမှက်နှာ	၁၁၂
17.	Pointers	စာမှက်နှာ	၁၄၀
18.	Header Files	စာမှက်နှာ	၁၄၉
19.	STL Containers	စာမှက်နှာ	၁၆၃
20.	Vector	စာမှက်နှာ	၁၇၂

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

21. List	စာမှက်နှာ	၁၈၆
22. Set	စာမှက်နှာ	၂၀၁
23. Map	စာမှက်နှာ	၂၁၆
24. Sorting Algorithms	စာမှက်နှာ	၂၃၂
25. Friend Keyword	စာမှက်နှာ	၂၄၇
26. Templates	စာမှက်နှာ	၂၅၉
27. File	စာမှက်နှာ	၂၆၉
28. Namespace	စာမှက်နှာ	၂၉၁
29. Thread	စာမှက်နှာ	၃၀၉
30. Games	စာမှက်နှာ	၃၂၂
31. Practical Projects	စာမှက်နှာ	၃၃၄
32. ကျေးဇူးတင်စကား	စာမှက်နှာ	၃၇၂
33. References	စာမှက်နှာ	၃၇၃
34. စာရေးသူသမိုင်းအကျဉ်း	စာမှက်နှာ	၃၇၄
35. စာရေးသူအိုင်တီးအတွေ့အကြုံမှတ်တမ်း	စာမှက်နှာ	၃၇၅
36. စာရေးသူ Documentary Photos	စာမှက်နှာ	၃၇၆

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

C++ HISTROY

❒ C++ သမိုင်းအကျဉ်း

C++ programming language ကို Bjarne Stroustrup ဆိုသူက 1980 ခုနှစ်မှာ စတင်ဖန်တီးခဲ့ပါတယ်။ သူဟာ Bell Labs (ယခု Nokia Bell Labs) မှာ အလုပ်လုပ်နေစဉ် C ဘာသာရပ်ကို အခြေခံပြီး ပိုမိုအဆင့်မြင့်တဲ့ ပရိုဂရမ်းမင်းဘာသာရပ်တစ်ခု ဖန်တီးဖို့ ကြိုးစားခဲ့ပါတယ်။

C++ ရဲ့ အစ Early Development တွေကို ပြောရမယ်ဆိုရင် 1979 - Stroustrup ဟာ "C with Classes" ဆိုတဲ့ ပရိုဂရမ်းမင်းဘာသာရပ်ကို စတင်ခဲ့ပါတယ်။ 1983 - အဲဒီဘာသာရပ်ကို "C++" လို့ အမည်ပြောင်းလိုက်ပါတယ်။ ++ ဆိုတာ C မှာ increment operator ဖြစ်ပြီး၊ C ထက် ပိုကောင်းတဲ့ ဘာသာရပ်လို့ အဓိပ္ပာယ်ရပါတယ်။ 1985 - The C++ Programming Language ဆိုတဲ့ ပထမဆုံး စာအုပ်ကို ထုတ်ဝေခဲ့ပါတယ်။ 1989 - C++ 2.0 ထွက်ရှိလာပြီး multiple inheritance, abstract classes စတဲ့ features တွေ ပါလာပါတယ်။

C++ ကို ISO/IEC (International Organization for Standardization) က 1998 မှာ ပထမဆုံး standard (ISO/IEC 14882:1998) အဖြစ် သတ်မှတ်ခဲ့ပါတယ်။ Version C++3 မှာ အနည်းငယ်ပြင်ဆင်မှုများနဲ့ update လုပ်ခဲ့တယ်။ Version C++11 ("Modern C++") - major update (auto, lambda, smart pointers စသည့် features များ) တွေပြုလုပ်ခဲ့တယ်။ C++14, C++17, C++20 - ဆက်လက် update တွေ လုပ်ပြီး ဒီနေအထိ ကျယ်ကျယ်ပြန့်ပြန့် အသုံးပြုနေဆဲပဲ ဖြစ်ပါတယ်။

C++ ဟာ high-performance systems တွေအတွက် အထူးသင့်တော်ပြီး၊ အဓိက Operating Systems (Windows, Linux, macOS တို့ရဲ့ အစိတ်အပိုင်းများ) ရေးသားဖန်တီးမှုတွေမှာ အသုံးပြုကြပါတယ်။ C++ နဲ့ Game Development (Unreal Engine, Game Engines) တွေ ၊ Embedded Systems (ကားများ၊ စက်ရုပ်များ၊ IoT devices) application တွေ ၊ High-Frequency

Trading (ငွေကြေးစျေးကွက် algorithms) application တွေ ရေးသားဖန်တီးမှုတွေမှာ အဓိက အသုံးများပါတယ်။

C++ ရဲ့ အဓိက အားသာချက်တွေကတော့ Speed ဖြစ်ပါတယ်။ C++ ဟာ machine code အဖြစ် တိုက်ရိုက် run နိုင်တာကြောင့် အလွန်မြန်ဆန်ပါတယ်။ ဒုတိယ အချက်ကတော့ Control ဖြစ်ပါတယ်။ Memory နဲ့ hardware resources တွေကို တိုက်ရိုက်ထိန်းချုပ်နိုင်ပါတယ်။ တတိယအချက်အနေနဲ့ ပြောရမယ်ဆိုရင်တော့ Object-Oriented - OOP concepts (classes, inheritance, polymorphism) တွေကို အားပေးပါတယ်။ နောက်ဆုံးအနေနဲ့ ပြောရမယ်ဆိုရင် Compatibility ဖြစ်ပါတယ်။ Python Programmer ပဲ ဖြစ်ဖြစ်၊ Java Programmer ပဲ ဖြစ်ဖြစ် customized application လုပ်တော့မယ်ဆိုရင် C++ ကို သုံးရပါတယ်။ C နဲ့ backward compatible ဖြစ်ပါတယ်။

Frist C++ Program

Code:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, C++ World!" << endl;
    return 0;
}
```

C++ ဟာ နှစ်ပေါင်း 40 ကျော် ကြာမြင့်ခဲ့ပြီဖြစ်ပေမယ့်၊ ယနေ့ထိတိုင် high-performance computing နဲ့ system programming မှာ အရေးပါနေဆဲဖြစ်ပါတယ်။

ဘာလို့ C++ ကို လေ့လာသင့်သလဲ?

❒ ဘာလို့ C++ ကိုလေ့လာသင့်သလဲ?

C++ ဟာ စွမ်းဆောင်ရည်မြင့်မားတဲ့ language တစ်ခုဖြစ်ပါတယ်။ C++ ဟာ system-level access ပေးနိုင်ပြီး hardware နဲ့ နီးကပ်စွာ အလုပ်လုပ်နိုင်ပါတယ်။ Game engines, operating systems နဲ့ database systems လို performance အရေးကြီးတဲ့ application တွေမှာ အဓိက အသုံးပြုပါတယ်။ Memory နဲ့ resource management တွေကို manual ထိန်းချုပ်နိုင်တာကြောင့် စွမ်းဆောင်ရည်မြင့်တဲ့ program တွေရေးနိုင်ပါတယ်။

C++ ဟာ versatile language တစ်ခုဖြစ်ပါတယ်။ Multi-paradigm language ဖြစ်တာကြောင့် object-oriented, procedural နဲ့ generic programming တွေကို ပေါင်းစပ်အသုံးပြုနိုင်ပါတယ်။ System software ကနေ game development အထိ နယ်ပယ်စုံမှာ အသုံးဝင်ပြီး cross-platform development အတွက်လည်း သင့်တော်ပါတယ်။

Career opportunities တွေများတာကိုလည်း ပြောပြချင်ပါတယ်။ Tech giants ကြီးတွေဖြစ်တဲ့ Google, Microsoft, Amazon တို့မှာ C++ developer တွေကို အမြဲလိုအပ်နေပါတယ်။ Game development studios တွေမှာလည်း Unreal Engine လို game engine တွေအတွက် C++ skills ကို လိုအပ်ပါတယ်။ Embedded systems, finance, automotive စတဲ့ နယ်ပယ်တွေမှာလည်း လစာကောင်းတဲ့ အလုပ်အခွင့်အလမ်းတွေ များစွာရှိပါတယ်။

Programming အခြေခံတွေကို နက်နက်နဲနဲနားလည်ချင်ရင် C++ ကို လေ့လာသင့်ပါတယ်။ C++ ဟာ memory management ကို နားလည်စေပြီး data structures နဲ့ algorithms တွေကို အတွင်းကျကျ သိရှိစေပါတယ်။ Computer architecture နဲ့ နီးကပ်စွာ အလုပ်လုပ်ရတာကြောင့် ကွန်ပျူတာအလုပ်လုပ်ပုံကို ပိုမိုနားလည်စေပါတယ်။ စာရေးသူက C++ ကို လေးနှစ်လောက် အချိန်ပေးပြီး ဆက်တိုက်လေ့လာခဲ့ပါတယ်။ အဲဒါကြောင့် ကျနော်က Programming ကို စိတ်ဝင်စားတဲ့သူ Prorgramming ကို စလုပ်မယ့် သူတွေနဲ့ စကားပြောဖြစ်တိုင်း ၊ သူတို့တွေ အကြံတောင်းတိုင် C++ ကို စလုပ်ဖို့ပဲ အမြဲ တိုက်တွန်းပါတယ်။ ယခုခေတ်အခါ မှာ တော့ Internet ရှိတာရယ် လေ့လာသင်ယူမှုတွေမှာ ကူညီနိုင်မဲ့ အိုင်တီကျမ်းကျင် programmer တွေအများကြီး ရှိနေတာကြောင့် ၂ နှစ်လောက် အချိန်ပေးရင် အဆင်ပြေမယ်လို့ ထင်ပါတယ်။

Modern technologies တွေနဲ့ compatibility ရှိတာကလည်း C++ ရဲ့အားသာချက်တစ်ခုပါ။ AI/ML frameworks တွေဖြစ်တဲ့ TensorFlow, PyTorch တို့ရဲ့ core တွေက C++ နဲ့ရေးထားပါတယ်။ Blockchain technologies နဲ့ IoT devices development တွေမှာလည်း C++ ကို အသုံးများပါတယ်။

Educational benefits အနေနဲ့ကတော့ C++ ဟာ computer science နဲ့ computer engineering ဘာသာရပ်တွေမှာ အခြေခံအနေနဲ့ သင်ကြားလေ့ရှိပါတယ်။ Competitive programming ပြိုင်ပွဲတွေမှာလည်း C++ ကို အသုံးများပြီး complex problem solving skills တွေ တိုးတက်စေပါတယ်။

နောက်ဆုံးအနေနဲ့ C++ ရဲ့ longevity ကို ပြောပြချင်ပါတယ်။ 1985 ခုနှစ်ကနေ စတင်ခဲ့တဲ့ language တစ်ခုဖြစ်ပြီး နှစ် 40 နီးပါး အသုံးပြုနေဆဲဖြစ်ပါတယ်။ ISO Standard ဖြစ်တာကြောင့် နှစ်စဉ် update တွေရရှိနေဆဲဖြစ်ပြီး backward compatibility လည်း ကောင်းမွန်ပါတယ်။ C++ ကို လေ့လာခြင်းဖြင့် သင်ဟာ programming နဲ့ပတ်သက်တဲ့ နက်နဲတဲ့အသိပညာတွေရရှိမှာဖြစ်ပြီး နည်းပညာလောကမှာ ရေရှည်အသုံးဝင်မယ့် skill တစ်ခုကို ပိုင်ဆိုင်ရမှာဖြစ်ပါတယ်။

System Requirements and Software Tools

❒ C++ System Requirements and Software Tools

C++ ကို အသုံးပြုရန် လိုအပ်သော System Requirements နှင့် Software Tools အကြောင်းကို ရှင်းပြပေးပါမယ်။

✓System Requirements (အခြေခံ စနစ် လိုအပ်ချက်များ)

C++ ကို ရိုးရိုး program များ ရေးသားရန် အတွက် အောက်ပါ အခြေခံ system လိုအပ်ချက်များ ရှိပါတယ်။

- Operating System (OS):
  - Windows (7/8/10/11)
  - Linux (Ubuntu, Fedora, Debian, etc.)
  - macOS
- Processor (CPU):
  - Modern CPU (Intel/AMD) – 1GHz နှင့်အထက်
- Memory (RAM):
  - အနည်းဆုံး 4GB (ကြီးမားသော project များအတွက် 8GB+)
- Storage (HDD/SSD):
  - 5GB လွတ်နေရန် လိုအပ်ပါတယ် (IDE နှင့် compiler အတွက်)

Software Tools (လိုအပ်သော ဆော့ဖ်ဝဲများ)

C++ ဖြင့် programming လုပ်ရန် အောက်ပါ tools များ လိုအပ်ပါတယ်။

1. Text Editors / IDEs (Code ရေးရန်)

- Visual Studio Code (VS Code) – Extensions (C++ extension, Code Runner) ဖြင့် အသုံးပြုနိုင်ပါတယ်။
- Code::Blocks – Lightweight IDE (Windows, Linux, macOS)

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

- Dev-C++ – Windows အတွက် ရိုးရှင်းသော IDE
- CLion – Professional C++ IDE (Paid)
- Xcode – macOS အတွက် Apple ၏ IDE

2. Compilers (Code ကို Machine Language ပြောင်းရန်)

- GCC (GNU Compiler Collection) – Linux/macOS တွင် အသုံးများပါတယ်။
- MinGW (Windows အတွက် GCC)
- Clang – Modern C++ compiler (macOS/Linux)
- Microsoft Visual C++ (MSVC) – Visual Studio နှင့် တွဲဖက် အသုံးပြုနိုင်ပါတယ်။

3. Build Tools (Project Management)

- CMake – Cross-platform build system
- Make – Linux/macOS တွင် အသုံးပြုသော build tool

4. Debuggers (Error ရှာဖွေရန်)

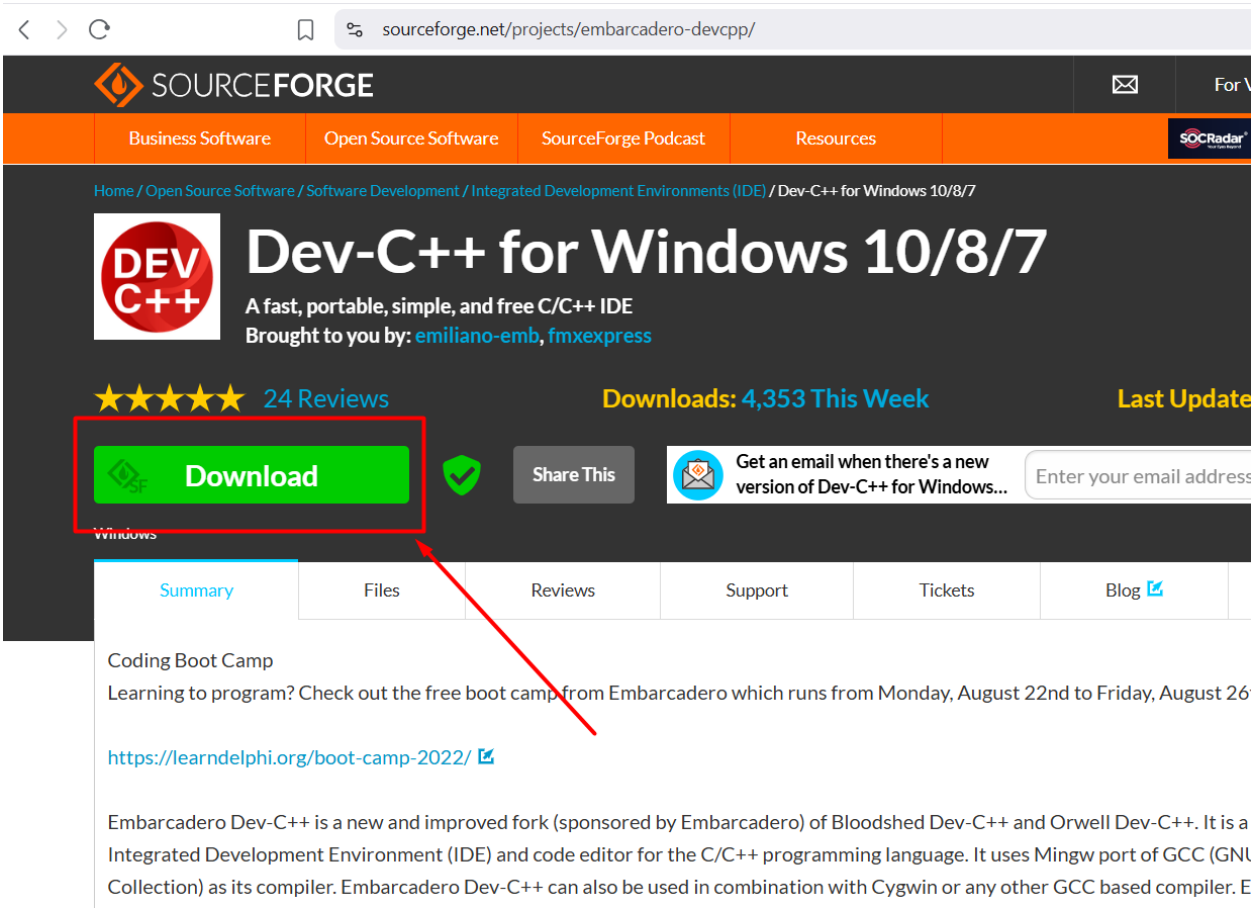
- GDB (GNU Debugger) – Linux/macOS
- LLDB – Clang-based debugger
- Visual Studio Debugger – Windows

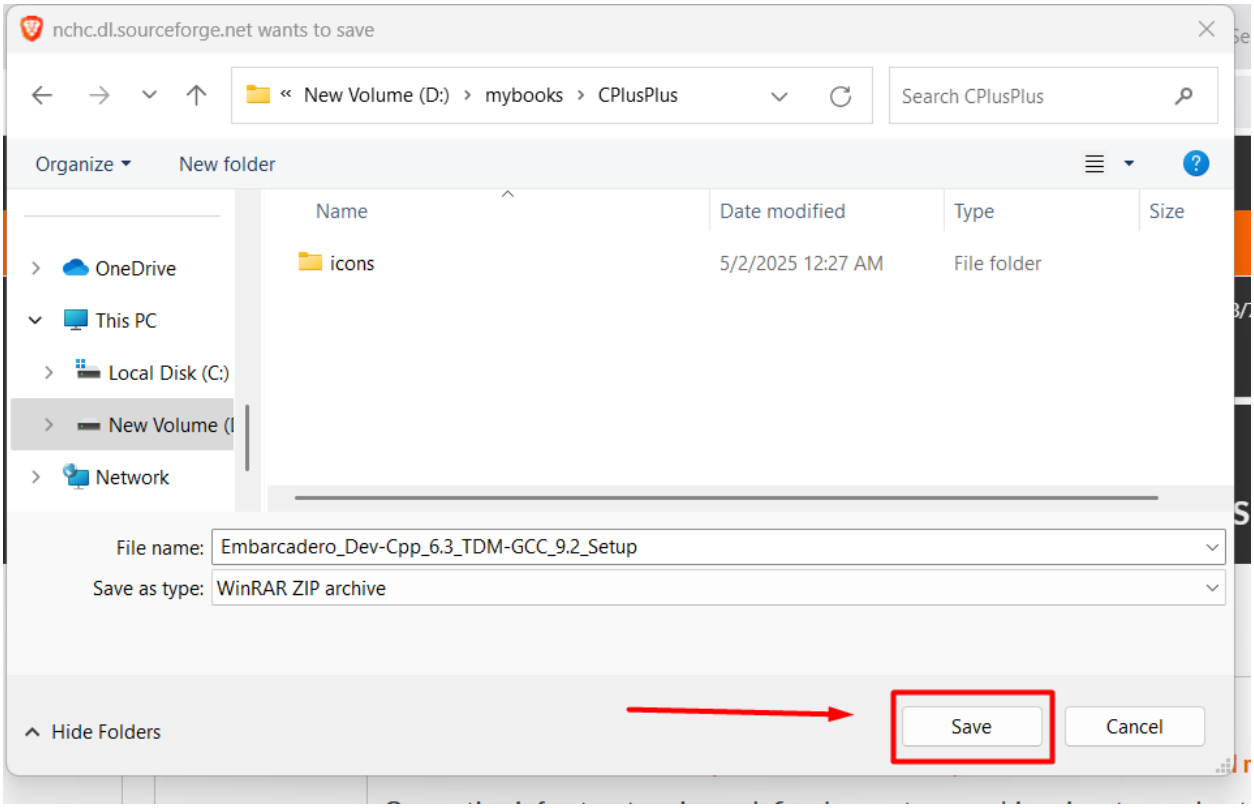
အချုပ်အားဖြင့် C++ ကို စတင်အသုံးပြုရန် Text Editor / IDE တစ်ခုနှင့် Compiler တစ်ခု လိုအပ်ပါတယ်။ Windows အတွက် VS Code + MinGW သို့မဟုတ် Dev-C++၊ Linux/macOS အတွက် GCC/Clang တို့ကို အသုံးပြုနိုင်ပါတယ်။ ဒီစာအုပ်ထဲမှာ Dev-C++ ကို အသုံးပြုထားပါတယ်။ သုံးရတာ အရမ်းကို ရိုးရှင်းပြီးတော့ လွယ်ကူပါတယ်။ အောက် က Download Link ကို ခေါက်ပြီး DevC++ compiler 6.3 ကို ရယူလိုက်ပါ။

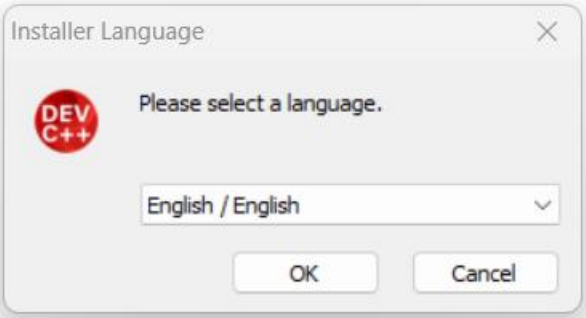
Download Link:

Dev-C++ for Windows 10/8/7 download | SourceForge.net

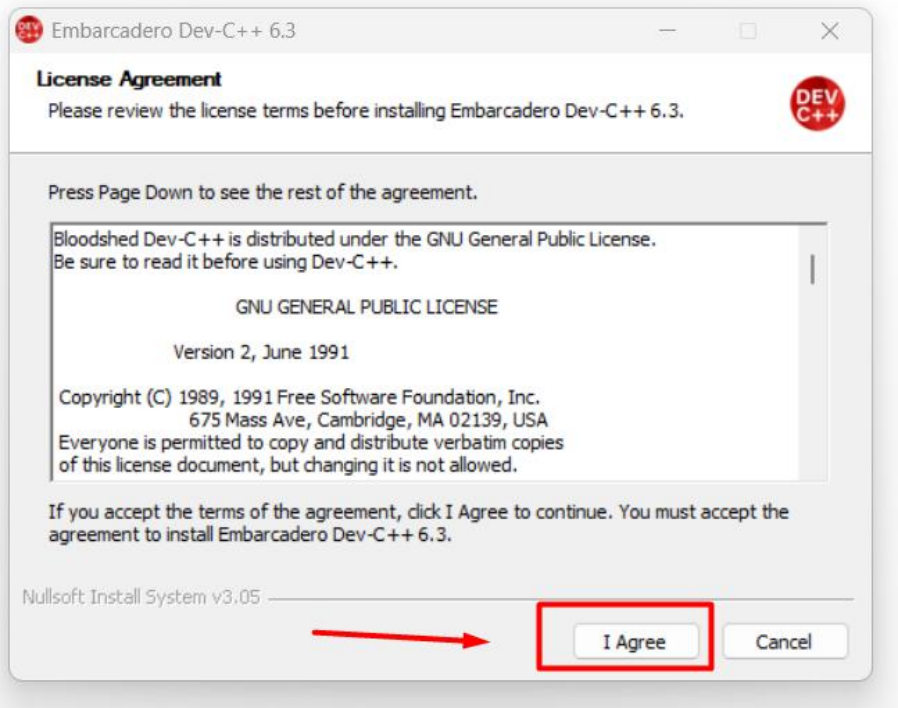
Installation အဆင့်ဆင့်ကို အောက်ပါအတိုင်း လိုက်လုပ်ပါ။

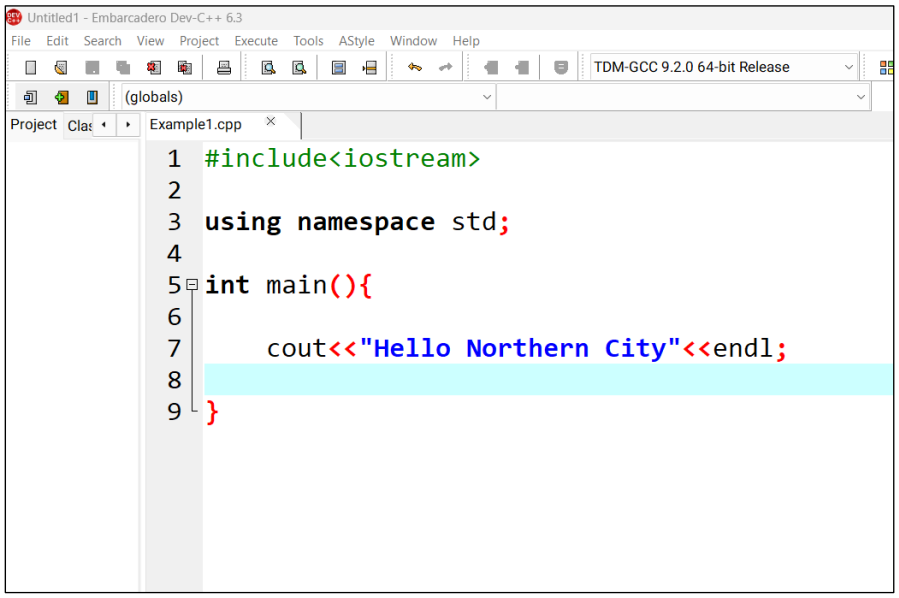

sourceforge.net/projects/embarcadero-devcpp/
SOURCEFORGE
Business Software
Open Source Software
SourceForge Podcast
Resources
Home / Open Source Software / Software Development / Integrated Development Environments (IDE) / Dev-C++ for Windows 10/8/7
Dev-C++ for Windows 10/8/7
A fast, portable, simple, and free C/C++ IDE
Brought to you by: emiliano-emb, fmxexpress
24 Reviews
Downloads: 4,353 This Week
Last Update
Download
Share This
Get an email when there's a new version of Dev-C++ for Windows...
Enter your email address
Summary
Files
Reviews
Support
Tickets
Blog
Coding Boot Camp
Learning to program? Check out the free boot camp from Embarcadero which runs from Monday, August 22nd to Friday, August 26
https://learndelphi.org/boot-camp-2022/
Embarcadero Dev-C++ is a new and improved fork (sponsored by Embarcadero) of Bloodshed Dev-C++ and Orwell Dev-C++. It is a
Integrated Development Environment (IDE) and code editor for the C/C++ programming language. It uses Mingw port of GCC (GNU
Collection) as its compiler. Embarcadero Dev-C++ can also be used in combination with Cygwin or any other GCC based compiler. E

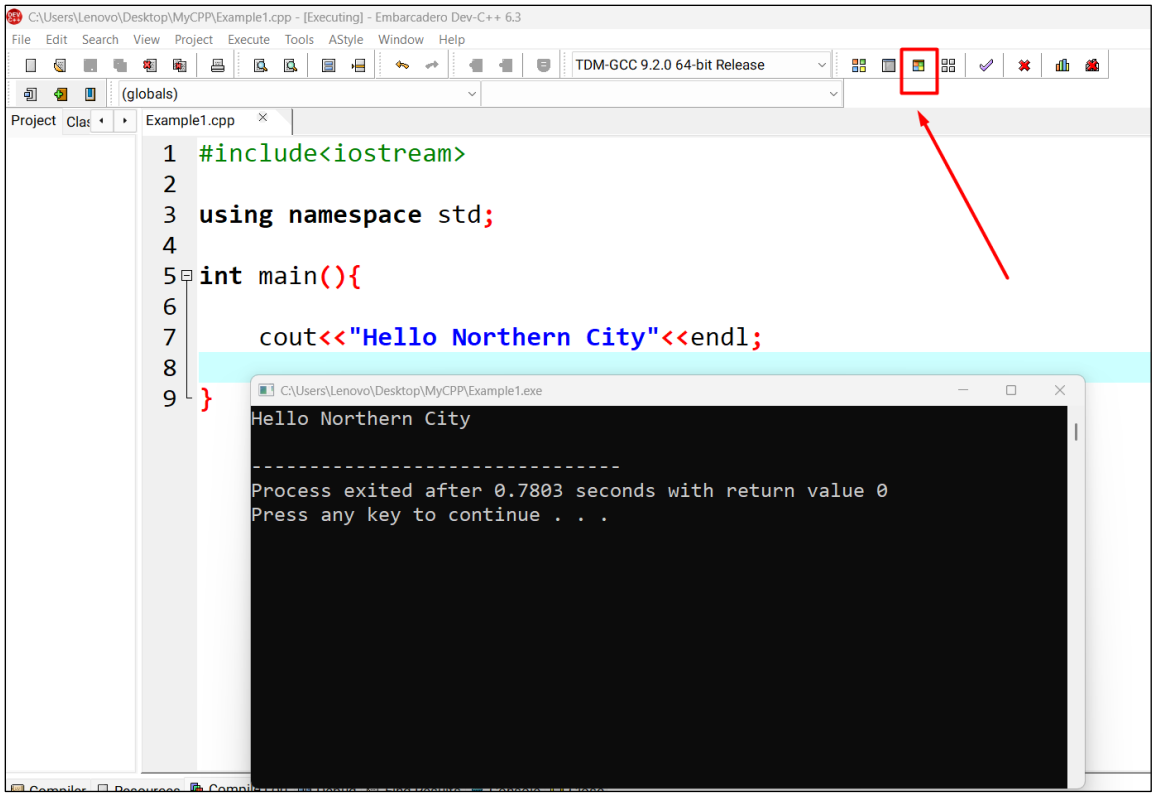




မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။


Embarcadero Dev-C++ 6.3
License Agreement
Please review the license terms before installing Embarcadero Dev-C++ 6.3.
Press Page Down to see the rest of the agreement.
Bloodshed Dev-C++ is distributed under the GNU General Public License.
Be sure to read it before using Dev-C++.
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Embarcadero Dev-C++ 6.3.
Nullsoft Install System v3.05
I Agree
Cancel


Untitled1 - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Project
Example1.cpp
#include<iostream>
using namespace std;
int main(){
cout<<"Hello Northern City"<<endl;
}


C:\Users\Lenovo\Desktop\MyCPP\Example1.cpp - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Project Clas
Example1.cpp
1 #include<iostream>
2
3 using namespace std;
4
5 int main(){
6
7 cout<<"Hello Northern City"<<endl;
8
9 }
C:\Users\Lenovo\Desktop\MyCPP\Example1.exe
Hello Northern City

Process exited after 0.7803 seconds with return value 0
Press any key to continue . . .

DATA TYPES

❒ Data Types

Data type (ဒေတာအမျိုးအစား) ဆိုတာ variable တစ်ခုမှာ သိမ်းဆည်းမယ့် data အမျိုးအစားကို သတ်မှတ်ပေးတာပါ။ C++ မှာ အဓိက data types တွေကို အောက်ပါအတိုင်း အုပ်စုခွဲနိုင်ပါတယ်။

1. Primitive Data Types (အခြေခံ ဒေတာအမျိုးအစားများ)

Integer Types (ကိန်းပြည့်များ)

- int - ပုံမှန် integer (4 bytes)

```
int age = 25;
```

- short - ပိုသေးတဲ့ integer (2 bytes)

```
short temperature = -10;
```

- long - ပိုကြီးတဲ့ integer (4 or 8 bytes)

```
long population = 8000000;
```

- long long - အကြီးဆုံး integer (8 bytes)

```
long long bigNumber = 9000000000;
```

Floating-Point Types (ဒသမကိန်းများ)

- float - single precision (4 bytes)

```
float pi = 3.14f;
```

- double - double precision (8 bytes)

```
double preciseValue = 3.1415926535;
```

- long double - extended precision (8, 12 or 16 bytes)

```
long double highPrecision = 3.141592653589793238L;
```

Character Types (အက္ခရာများ)

- char - single character (1 byte)

```
char grade = 'A';
```

- wchar_t - wide character (2 or 4 bytes)

```
wchar_t wideChar = L'a';
```

Boolean Type (အတွက်အချက်)

- bool - true/false (1 byte)

```
bool isPassed = true;
```

2. Derived Data Types (ဆင်းသက်လာသော ဒေတာအမျိုးအစားများ)

- array - တူညီတဲ့ data type ရှိတဲ့ value တွေကို စုစည်းထားခြင်း

```
int numbers[5] = {1, 2, 3, 4, 5};
```

- pointer - memory address ကို သိမ်းဆည်းထားခြင်း

```
int* ptr = &age;
```

- reference - variable တစ်ခုကို အခြား name တစ်ခုနဲ့ ရည်ညွှန်းခြင်း

```
int &ref = age;
```

3. User-Defined Data Types (အသုံးပြုသူသတ်မှတ်ထားသော ဒေတာအမျိုးအစားများ)

- struct - different data types တွေကို စုစည်းထားခြင်း

```
struct Student {
    string name;
    int age;
    float gpa;
```

```
};
```

- class - data နဲ့ functions တွေကို ထုပ်ပိုးထားခြင်း

```
class Car {
    public:
        string model;
        int year;
};
```

- union - memory location တစ်ခုတည်းကို different data types တွေနဲ့ share လုပ်ခြင်း

```
union Data {
    int i;
    float f;
};
```

- enum - named integer constants တွေကို သတ်မှတ်ခြင်း

```
enum Color { RED, GREEN, BLUE };
```

4. Type Modifiers (အမျိုးအစား ပြုပြင်ရေးသတ်မှတ်ချက်များ)

- signed - အနုတ်နဲ့ အပေါင်း နှစ်မျိုးလုံး (default)

```
signed int s = -10;
```

- unsigned - အပေါင်းကိန်းများသာ

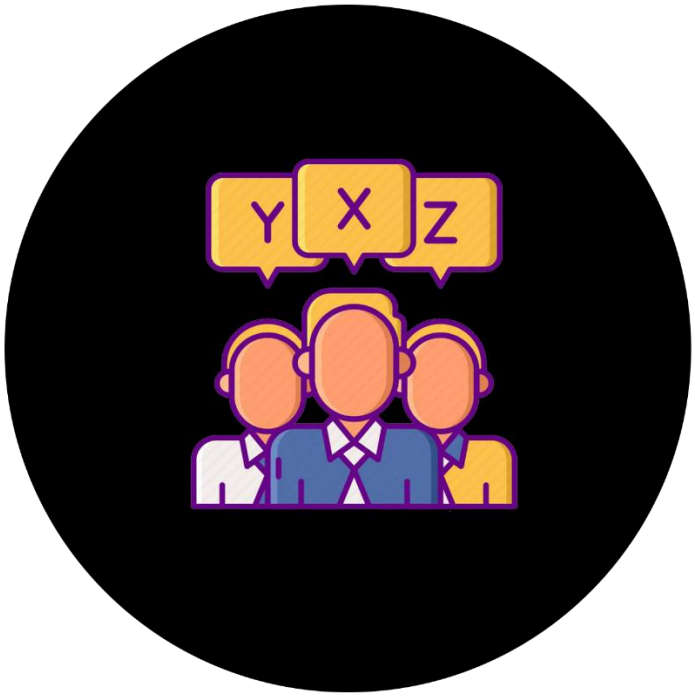
```
unsigned int u = 100;
```

❒ Size နဲ့ Range များ

Data type တစ်ခုချင်းစီမှာ သိမ်းဆည်းနိုင်တဲ့ size နဲ့ range ကွာခြားပါတယ်။

Data Type	Size (bytes)	Range
int	4	-2,147,483,648 to 2,147,483,647
float	4	1.2E-38 to 3.4E+38
double	8	2.3E-308 to 1.7E+308
char	1	-128 to 127 or 0 to 255

Variables



❒ Variables

Variable ဆိုတာ program တစ်ခုမှာ data တွေကို သိမ်းဆည်းဖို့ နေရာလွတ်တစ်ခုပါ။ variable တစ်ခုကို သုံးမယ်ဆိုရင် သူ့ကို အရင်ဆုံး declare (ကြေငြာမယ်) လုပ်ဖို့ လိုပါတယ်။

❒ Variable Declaration Syntax

```
data_type variable_name;
```

ဥပမာ:

```
int age;        // integer variable
double salary;  // decimal variable
char grade;     // character variable
string name;    // string variable
```

❒ Variable Initialization (တန်ဖိုးစတင်သတ်မှတ်ခြင်း)

✓ Variable ကို declare လုပ်တဲ့အချိန်မှာပဲ တန်ဖိုးစသတ်မှတ်နိုင်ပါတယ်။

```
int count = 10;
double pi = 3.14159;
char letter = 'A';
string greeting = "Hello";
```

❒ Variable Naming Rules

1. Letter, underscore(_) သို့မဟုတ် number တွေနဲ့ စနိုင်ပါတယ် (ဒါပေမယ့် number နဲ့ မစရပါ)

2. C++ reserved keywords တွေကို variable name အဖြစ် မသုံးရပါ (ဥပမာ: int, double, return စသည်)
3. Case-sensitive ဖြစ်ပါတယ် (age နဲ့ Age က တူတဲ့ variable မဟုတ်ပါ)

❒ ကောင်းသော variable names များ:

```
int user_age;
double accountBalance;
string firstName;
```

❒ မကောင်းသော variable names များ:

```
int 123age;     // number နဲ့စနေသည်
double double;  // keyword ကိုသုံးထား
char first name; // space ပါနေသည်
```

❒ Variable Types (*variable* အမျိုးအစားများ)

✓ C++ မှာ အဓိက variable types တွေကတော့:

- int - integer numbers (ဥပမာ: 10, -5, 0)
- float/double - floating point numbers (ဥပမာ: 3.14, -0.001)
- char - single character (ဥပမာ: 'A', 'b', '1')
- string - text (ဥပမာ: "Hello", "John")
- bool - boolean (true/false)

ဥပမာ Program:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    int student_age = 20;
    double average_score = 85.5;
    char class_section = 'B';
    string university = "Computer University";

    cout << "Age: " << student_age << endl;
    cout << "Score: " << average_score << endl;
    cout << "Section: " << class_section << endl;
    cout << "University: " << university << endl;

    return 0;
}
```

❒ Variable Naming Conventions

Programming Variable Naming Cases အကြောင်း ရှင်းပြချက်

Variable တွေကို နာမည်ပေးတဲ့အခါ Naming Conventions (အမည်ပေးစည်းမျဉ်းများ) ကို လိုက်နာဖို့ အရေးကြီးပါတယ်။ ဒီစည်းမျဉ်းတွေက code ကို ရှင်းလင်းစွာ ဖတ်ရှုနိုင်အောင် နဲ့ တသမတ် တည်း ရှိအောင် ကူညီပေးပါတယ်။

1. Camel Case

- ပထမစာလုံးကို စာသေးနဲ့စပြီး နောက်စာလုံးတိုင်းရဲ့ ပထမစာလုံးကို စာကြီးနဲ့ ရေးပါတယ်။
- Java, JavaScript, C# တို့မှာ အသုံးများပါတယ်။

ဥပမာ:

```
int userAge = 25;
string firstName = "John";
double accountBalance = 1000.50;
```

---

2. Pascal Case

- စာလုံးတိုင်းရဲ့ ပထမစာလုံးကို စာကြီးနဲ့ ရေးပါတယ်။
- Class, Struct, Interface တွေကို နာမည်ပေးတဲ့အခါ အသုံးများပါတယ် (C++, C#, Java)။

ဥပမာ:

```
class StudentProfile { ... };
struct BankAccount { ... };
string FullName = "Alice";
```

---

3. Snake Case

- စာလုံးတွေကြားမှာ underscore (_) ထည့်ပြီး အားလုံးကို စာသေးနဲ့ ရေးပါတယ်။
- Python, C, C++ (အချို့ standard libraries) တို့မှာ အသုံးများပါတယ်။

ဥပမာ:

```
int user_age = 30;
string first_name = "Bob";
double account_balance = 500.75;
```

---

4. Kebab Case

- စာလုံးတွေကြားမှာ hyphen (-) ထည့်ပြီး အားလုံးကို စာသေးနဲ့ ရေးပါတယ်။
- HTML, CSS, URL တို့မှာ အသုံးများပြီး variable names အတွက် C++, Java, Python တို့မှာ မသုံးပါ (ဘာကြောင့်လဲဆိုတော့ hyphen ကို operator အဖြစ် သတ်မှတ်ထားလို့ပါ)။

ဥပမာ (HTML/CSS):

```
<div class="user-profile"></div>
```

```
.font-size { ... }
```

---

5. UPPER CASE (စာကြီးအားလုံး)

- Constant (မပြောင်းလဲနိုင်သော) variables တွေကို သတ်မှတ်တဲ့အခါ အသုံးပြုပါတယ်။
- C, C++, Python, Java စတဲ့ ဘာသာရပ်တွေမှာ အသုံးများပါတယ်။

ဥပမာ:

```
const int MAX_USERS = 100;
const double PI = 3.14159;
```

✓ဘယ် Case ကို ဘယ်နေရာမှာ သုံးမလဲ?

Case Style	အသုံးပြုသည့်နေရာ	ဥပမာ
Camel Case	variables, functions (Java, JavaScript)	userAge, getUserName()
Pascal Case	classes, structs (C++, C#, Java)	StudentProfile, BankAccount
Snake Case	variables, functions (Python, C)	user_age, calculate_total()
Kebab Case	HTML/CSS classes, IDs	main-header, font-size
UPPER CASE	constants (အားလုံး)	MAX_SIZE, PI

ကောင်းမွန်တဲ့ Variable Names ရေးနည်း

1. ရှင်းလင်းပြီး အဓိပ္ပာယ်ရှိရမယ်
   - ❌ int a = 10; (မကောင်း)
   - ✅ int age = 25; (ကောင်း)
2. သင့်တော်တဲ့ Case Style ကို ရွေးချယ်ရမယ်
   - C++/Java → camelCase or PascalCase
   - Python → snake_case

3. Number နဲ့ မစပါနဲ့

- ❌ int 1stPlace; (မှား)
- ✅ int firstPlace; (မှန်)

4. *Special Characters (&, , -, space) မသုံးပါနဲ့*

- ❌ string user-name; (မှား)
- ✅ string userName; or string user_name; (မှန်)

❐ C++ တွင် variable တွေကို မှန်ကန်စွာ အသုံးပြုပုံ

```
#include <iostream>
using namespace std;

const int MAX_SCORE = 100;  // UPPER_CASE for constants

class PlayerProfile {       // PascalCase for class
private:
    string playerName;      // camelCase for member variables
    int highScore;

public:
    void set_player_name(string name) {  // snake_case for functions (optional
)
        playerName = name;
    }
};

int main() {
    int current_score = 95;  // snake_case (or camelCase)
    cout << "Max Score: " << MAX_SCORE << endl;
    return 0;
}
```

✓ Short Notes:

- C++ မှာ Camel Case (userName) နဲ့ Pascal Case (ClassName) ကို အဓိက အသုံးပြုပါတယ်။
- Python မှာ Snake Case (user_name) ကို ပိုသုံးပါတယ်။
- HTML/CSS မှာ Kebab Case (main-header) ကို သုံးပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

BASIC CALCULATIONS

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ Basic Calculations

1. Addition

```
#include <iostream>
using namespace std;

int main() {
    int input1, input2;

    cout << "Enter Input1:";
    cin >> input1;

    cout << "Enter Input2:";
    cin >> input2;

    int answer = input1 + input2;

    cout << "The answer is :" << input1 << " + " << input2 << " = " << answer
<< endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. int input1, input2; ဆိုတာက integer ကိန်းတွေ သိမ်းဖို့ variable နှစ်ခု ကြေငြာထားတာ ဖြစ်ပါတယ်။
2. cin >> input1; ဆိုတာက user ထည့်လိုက်တဲ့ တန်ဖိုးကို input1 variable ထဲ သိမ်းတာ ဖြစ်ပါတယ်။
3. answer variable ထဲမှာ နှစ်ခုပေါင်းထားတဲ့ တန်ဖိုးကို သိမ်းထားပါတယ်။
4. cout နဲ့ ရလဒ်ကို ပြသထားပါတယ်။

output:

```
Enter Input1: 10
Enter Input2: 5
The answer is : 10 + 5 = 15
```

2. Subtraction

```
#include <iostream>
using namespace std;

int main() {
    int input1, input2;


    cout << "Enter Input1:";
    cin >> input1;

    cout << "Enter Input2:";
    cin >> input2;

    int answer = input1 - input2;

    cout << "နုတ်လဒ်:" << input1 << " - " << input2 << " = " << answer << endl;


    return 0;
}
```

✓ ရှင်းလင်းချက်

1. အပေါ်က ဥပမာနဲ့ အတူတူပါပဲ၊ ဒါပေမယ့် ဒီမှာတော့ နုတ်တဲ့ operation ကို လုပ်ထားတာ ဖြစ်ပါတယ်။
2. answer variable ထဲမှာ ပထမဂဏန်းကနေ ဒုတိယဂဏန်းကို နုတ်ထားတဲ့ တန်ဖိုး သိမ်းထားပါတယ်။

output:

```
Enter Input1: 10
Enter Input2: 5
The answer is : 10 - 5 = 5
```

3. ပေါင်းခြင်း၊ နုတ်ခြင်း

```
#include <iostream>
using namespace std;

int main() {
    int input1, input2;

    cout << "Enter Inpu1: ";
    cin >> input1;

    cout << "Enter Input2: ";
    cin >> input2;

    int sum = input1 + input2;
    int difference = input1 - input2;
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
    cout << "Add answer is :" << input1 << " + " << input2 << " = " << sum <<
endl;
    cout << "Subtract answer is :" << input1 << " - " << input2 << " = " << di
fference << endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. ဒီဥပမာမှာတော့ တစ်ခါတည်း နှစ်မျိုးလုံးကို တွက်ပြထားတာ ဖြစ်ပါတယ်။
2. sum variable မှာ ပေါင်းလဒ်၊ difference variable မှာ နုတ်လဒ်ကို သိမ်းထားပါတယ်။
3. ရလဒ်နှစ်ခုလုံးကို အသီးသီး ပြသထားပါတယ်။

Output:

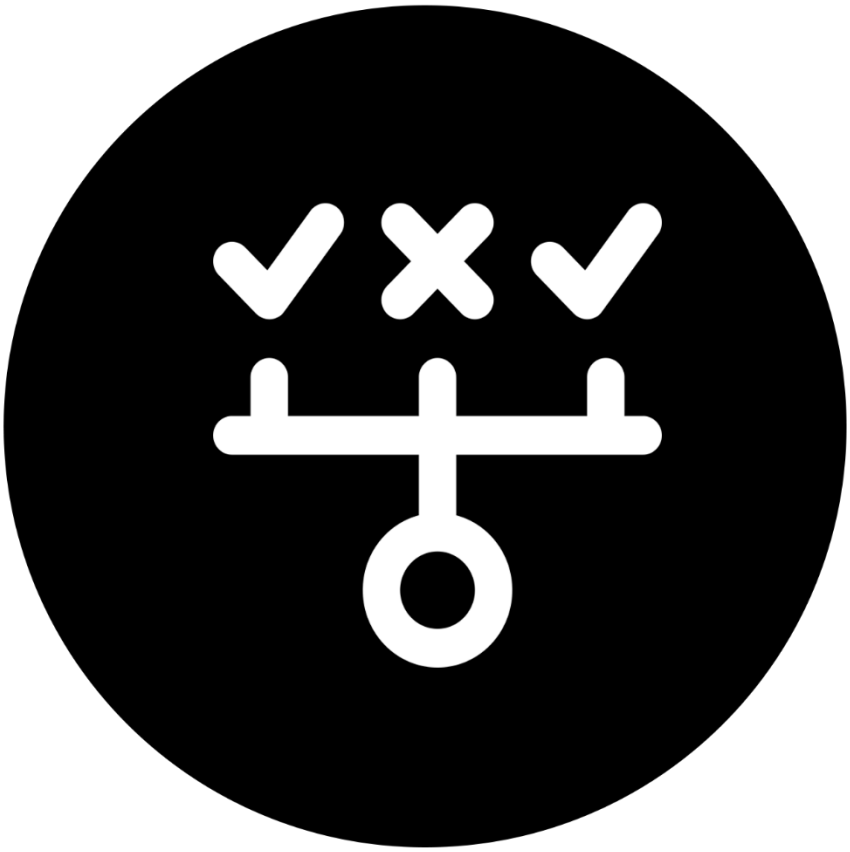
```
Enter Input1: 10
Enter Input2: 5
Add Answer is : 10 + 5 = 15
Subtract Answer is : 10 - 5 = 5
```

အရေးကြီးသော အချက်များ:

1. cin ကို သုံးပြီး user input ကို ဖတ်ယူနိုင်ပါတယ်။
2. Variable တွေကို သုံးမယ်ဆိုရင် အရင် ကြေငြာ(declare) ရပါမယ်။
3. >> operator ကို သုံးပြီး input stream ကနေ data တွေ ဖတ်ယူပါတယ်။
4. ဒီဥပမာတွေကို ကူးပြီး ကိုယ်တိုင် စမ်းကြည့်ရင် ပိုပြီး နားလည်လာမယ် ဖြစ်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

CONDITIONAL STATEMENTS



❒ Conditional Statements

Conditional statements တွေက program တွေမှာ အခြေအနေတစ်ခုပေါ်မူတည်ပြီး မတူညီတဲ့ code တွေကို run ဖို့ သုံးပါတယ်။

1. if Statement

```
#include <iostream>
using namespace std;

int main() {
    int age;
    cout << "Enter your age";
    cin >> age;

    if (age >= 18) {
        cout << "You are eligible to vote: " << endl;
    }

    return 0;
}
```

✓ ရှင်းလင်းချက်

- if statement က condition မှန်ရင် (true ဖြစ်ရင်) အတွင်းက code တွေကို run မယ်
- ဒီဥပမာမှာ အသက် 18 နှစ် သို့မဟုတ် ပိုကြီးရင် message ပြမယ်

2. if-else Statement (အခြေအနေနှစ်မျိုးအတွက်)

```
#include <iostream>
using namespace std;
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
int main() {
    int number;
    cout << "Enter a number:";
    cin >> number;

    if (number % 2 == 0) {
        cout << "This is even number…" << endl;
    } else {
        cout << "This is odd number…" << endl;
    }

    return 0;
}
```

✓ ရှင်းလင်းချက်

- if condition မှန်ရင် ပထမ block ကို run မယ်
- မှားရင် else block ကို run မယ်
- % operator က ဘယ်နှစ်ခါပြတ်ဝင်လဲဆိုတာ ရှာတာ (remainder)

3. else-if Statement (အခြေအနေများစွာအတွက်)

```
#include <iostream>
using namespace std;

int main() {
    int score;
    cout << "Enter your score(0-100): ";
    cin >> score;

    if (score >= 90) {
        cout << "Grade A" << endl;
```

```
    } else if (score >= 80) {
        cout << "Grade B" << endl;
    } else if (score >= 70) {
        cout << "Grade C" << endl;
    } else if (score >= 60) {
        cout << "Grade D" << endl;
    } else {
        cout << "Grade F" << endl;
    }

    return 0;
}
```

✓ ရှင်းလင်းချက်:

- အဆင့်ဆင့် condition တွေကို စစ်ဆေးတာ
- ပထမ condition မှန်ရင် သူ့ block run ပြီး ကျန်တာတွေကို ကျော်သွားမယ်
- တစ်ခုမှ မမှန်ရင် else block ကို run မယ်

4. switch Statement

```
#include <iostream>
using namespace std;

int main() {
    char grade;
    cout << "Enter Your Grade(A,B,C,D,F):";
    cin >> grade;

    switch(grade) {
        case 'A':
            cout << "Excellent!..." << endl;
            break;
        case 'B':
```

```
            cout << "Very Good…" << endl;
            break;
        case 'C':
            cout << "Fair.." << endl;
            break;
        case 'D':
            cout << "Weak…" << endl;
            break;
        case 'F':
            cout << "Fail…" << endl;
            break;
        default:
            cout << "Invalid Grade…" << endl;
    }

    return 0;
}
```

✓ ရှင်းလင်းချက်

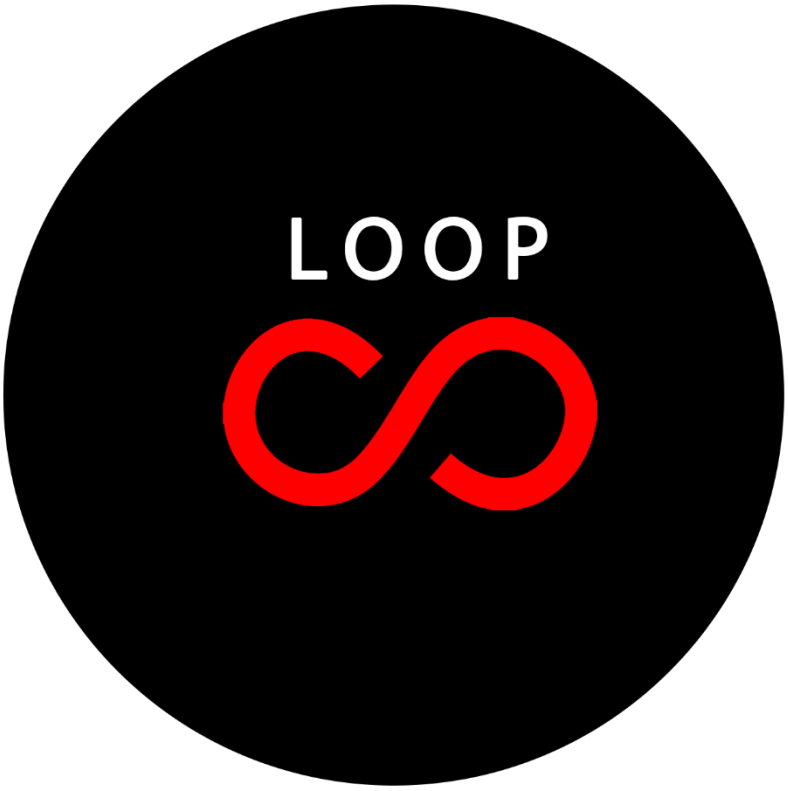
- switch က variable တစ်ခုရဲ့ တန်ဖိုးကို စစ်ဆေးတာ
- case တစ်ခုနဲ့ ကိုက်ညီရင် အဲ့ code block ကို run မယ်
- break မပါရင် နောက် case ကိုပါ ဆက်သွားမယ် (fall through)
- default က case တစ်ခုမှ မကိုက်ညီရင် run မယ်

အရေးကြီးသော အချက်များ:

- Condition တွေကို parentheses () ထဲမှာ ရေးရမယ်
- Code blocks တွေကို curly braces {} နဲ့ ဝန်းရံထားရမယ်
- == က equality ကို စစ်တာ၊ = က assignment လုပ်တာ (မှားတတ်တဲ့ အမှား)
- Logical operators တွေ (&&, ||, !) သုံးပြီး condition တွေကို ပေါင်းစပ်နိုင်တယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

LOOPS



❒ Loops

Loops တွေက code တွေကို အကြိမ်ကြိမ်အထပ်ထပ် run ဖို့ သုံးပါတယ်။ C++ မှာ အဓိက loop ၃ မျိုးရှိပါတယ်။ Loop မှာ အခြေခံအားဖြင့် ၃ မျိုးရှိပါတယ်။ အဲဒါတွေကတော့ -

1. For Loop
2. While Loop
3. Do While loop

တွေပဲဖြစ်ပါတယ်။

1. for Loop (အကြိမ်ရေသတ်မှတ်ပြီး လုပ်ဆောင်ခြင်း)

```
#include <iostream>
using namespace std;

int main() {
    // 1 ကနေ 10 အထိ ဂဏန်းတွေကို ရိုက်ထုတ်မယ်
    for(int i = 1; i <= 10; i++) {
        cout << i << " ";
    }

    return 0;
}
```

✓ ရှင်းလင်းချက်

- for loop မှာ အပိုင်း ၃ ပိုင်းပါဝင်ပါတယ်:
    1. int i = 1 - loop counter ကို စတင်တာ (initialization)
    2. i <= 10 - condition စစ်ဆေးတာ (ဘယ်အထိ run မလဲ)

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

3. i++ - loop counter ကို တိုးပေးတာ (increment)

- ဒီဥပမာမှာ 1 ကနေ 10 အထိ ဂဏန်းတွေကို ရိုက်ထုတ်ပါတယ်

✓ ရလဒ်

```
1 2 3 4 5 6 7 8 9 10
```

2. while Loop (အခြေအနေမှန်သရွေ့ လုပ်ဆောင်ခြင်း)

```
#include <iostream>
using namespace std;

int main() {
    int count = 1;

    // count က 5 ထက် နည်းနေသရွေ့ loop ကို ဆက်လုပ်မယ်
    while(count <= 5) {
        cout << "Hello World! (" << count << ")" << endl;
        count++;
    }

    return 0;
}
```

✓ ရှင်းလင်းချက်

- while loop က condition မှန်နေသရွေ့ ဆက်လုပ်သွားမယ်
- count++ ကို မမေ့ပါနဲ့၊ ဒါမှမဟုတ် infinite loop ဖြစ်သွားမယ်
- ဒီဥပမာမှာ "Hello World!" ကို 5 ခါ ရိုက်ထုတ်ပါတယ်

Output:

```
Hello World! (1)
Hello World! (2)
Hello World! (3)
Hello World! (4)
Hello World! (5)
```

3. do-while Loop (အနည်းဆုံး ၁ ကြိမ်တော့ လုပ်ဆောင်ခြင်း)

```
#include <iostream>
using namespace std;

int main() {
    int number;

    do {
        cout << "1 ကနေ 10 အထိ ကိန်းတစ်ခုထည့်ပါ: ";
        cin >> number;
    } while(number < 1 || number > 10);

    cout << "သင့်ရဲ့ရွေးချယ်မှု: " << number << endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

- do-while loop က အနည်းဆုံး ၁ ကြိမ်တော့ run မယ်
- condition ကို loop ရဲ့နောက်ဆုံးမှာ စစ်ဆေးတာ
- ဒီဥပမာမှာ user က 1-10 အတွင်းကိန်း မရမချင်း ထပ်မေးမယ်

✓ အထူး Loop Control Statements

1. break Statement (Loop ကနေ ရုတ်တရက်ထွက်ခြင်း)

```
#include <iostream>
using namespace std;

int main() {
    for(int i = 1; i <= 10; i++) {
        if(i == 5) {
            break;  // i က 5 ရောက်ရင် loop ကနေ ထွက်သွားမယ်
        }
        cout << i << " ";
    }

    return 0;
}
```

2. continue Statement (ဒီတစ်ကြိမ်တည်းကို skip လုပ်ခြင်း)

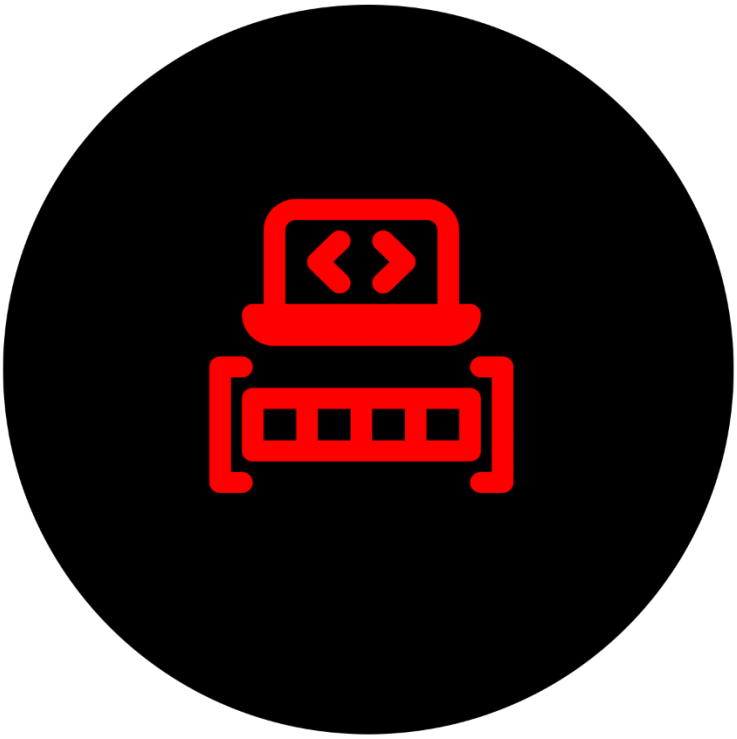
```
#include <iostream>
using namespace std;

int main() {
    for(int i = 1; i <= 10; i++) {
        if(i % 2 == 0) {
            continue;  // စုံကိန်းဆို skip လုပ်မယ်
        }
        cout << i << " ";
    }

    return 0;
}
```

✓ Short Notes:

1. Loop တစ်ခုစီမှာ initialization, condition နဲ့ increment/decrement ပါဝင်ဖို့ လိုပါတယ်
2. Infinite loop မဖြစ်အောင် ဂရုစိုက်ပါ (condition က အမြဲ true ဖြစ်နေရင်)
3. Nested loops (loop ထဲမှာ loop) တွေလည်း လုပ်လို့ရပါတယ်
4. break နဲ့ continue တို့ကို သင့်တော်သလို သုံးနိုင်ပါတယ်

Loops တွေကို ကျွမ်းကျင်စွာ အသုံးပြုနိုင်ရင် program တွေကို ပိုမိုထိရောက်စွာ ရေးသားနိုင်မယ် ဖြစ်ပါတယ်။

Arrays



❒ Arrays

Static arrays တွေက C++ မှာ အခြေခံကျတဲ့ data structure တစ်ခုဖြစ်ပြီး fixed size ရှိတဲ့ elements တွေကို သိမ်းဆည်းဖို့ သုံးပါတယ်။

1. Static Array ကြေငြာခြင်းနှင့် အသုံးပြုခြင်း

```
#include <iostream>
using namespace std;

int main() {
    // Array တစ်ခုကို ကြေငြာပြီး initialize လုပ်ခြင်း
    int numbers[5] = {10, 20, 30, 40, 50};

    // Array elements တွေကို access လုပ်ခြင်း
    cout << "ပထမအကြိမ်:" << endl;
    for(int i = 0; i < 5; i++) {
        cout << "numbers[" << i << "] = " << numbers[i] << endl;
    }

    // Array elements တွေကို ပြောင်းလဲခြင်း
    numbers[2] = 100;  // တတိယ element ကို 100 အဖြစ် ပြောင်းမယ်

    cout << "\nဒုတိယအကြိမ် (တတိယ element ပြောင်းပြီး):" << endl;
    for(int i = 0; i < 5; i++) {
        cout << "numbers[" << i << "] = " << numbers[i] << endl;
    }

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. int numbers[5] ဆိုတာက integer type 5 ခုပါတဲ့ array တစ်ခုကို ကြေငြာတာဖြစ်ပါတယ်
2. Array index က 0 ကနေ စပါတယ် (numbers[0] က ပထမ element)
3. Array size က compile time မှာ သိရတဲ့အတွက် ပြောင်းလဲလို့ မရပါဘူး
4. Loop သုံးပြီး array elements တွေကို အလွယ်တကူ access လုပ်နိုင်ပါတယ်

Output:

ပထမအကြိမ်:

```
numbers[0] = 10
numbers[1] = 20
numbers[2] = 30
numbers[3] = 40
numbers[4] = 50
```

ဒုတိယအကြိမ် (တတိယ `element` ပြောင်းပြီး):

```
numbers[0] = 10
numbers[1] = 20
numbers[2] = 100
numbers[3] = 40
numbers[4] = 50
```

2. Array Initialization နည်းလမ်းများ

```
#include <iostream>
using namespace std;

int main() {
    // နည်းလမ်း ၁: Size နဲ့ values တွေကို တစ်ပါတည်း ထည့်ခြင်း
    int arr1[3] = {1, 2, 3};
```

```
    // နည်းလမ်း ၂: Size ကို compiler က အလိုအလျောက် တွက်ချက်ခိုင်းခြင်း
    int arr2[] = {10, 20, 30, 40};  // size 4 array

    // နည်းလမ်း ၃: Size ပဲထားပြီး values တွေကို နောက်မှ ထည့်ခြင်း
    int arr3[5];
    arr3[0] = 5;
    arr3[1] = 10;
    // ... အခြား elements တွေက default 0 ဖြစ်သွားမယ်

    cout << "arr3[4] = " << arr3[4] << endl;  // 0 ဖြစ်မယ်

    return 0;
}
```

3. Array Size ရှာဖွေခြင်း

```
#include <iostream>
using namespace std;

int main() {
    float prices[] = {12.5, 7.8, 9.9, 15.2};

    // sizeof operator ကို သုံးပြီး array size ရှာမယ်
    int size = sizeof(prices) / sizeof(prices[0]);

    cout << "Array size: " << size << endl;
    cout << "Total bytes: " << sizeof(prices) << endl;
    cout << "One element size: " << sizeof(prices[0]) << endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. sizeof(prices) က array ရဲ့ total bytes ကို ပြန်ပေးမယ်
2. sizeof(prices[0]) က single element ရဲ့ size ကို ပြန်ပေးမယ်
3. ဒီနည်းနဲ့ array size ကို တွက်လို့ရတယ်

4. Multi-dimensional Arrays (2D Arrays)

```
#include <iostream>
using namespace std;

int main() {
    // 2D array (3 rows x 4 columns)
    int matrix[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    // Nested loops သုံးပြီး 2D array ကို access လုပ်မယ်
    cout << "2D Array Elements:" << endl;
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 4; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

✓ Short Notes:

1. Static arrays တွေက fixed size ရှိပြီး compile time မှာ memory allocate လုပ်ပါတယ်
2. Array index က 0 ကနေ စပါတယ် (0 to size-1)
3. Array bounds ကို မကျော်ပါနဲ့ (undefined behavior ဖြစ်နိုင်တယ်)
4. Arrays တွေကို functions ထဲကို pass by reference အနေနဲ့ ပို့ရပါတယ်
5. C-style strings တွေကလည် char arrays တွေပဲဖြစ်ပါတယ်

Static arrays တွေကို သေချာနားလည်ထားရင် C++ programming မှာ အရမ်းအသုံးဝင်ပါတယ်။ ဒါပေမယ့် size ကြိုတင်မသိတဲ့ အခြေအနေတွေအတွက် std::vector လို dynamic containers တွေကို သုံးဖို့လည်း စဉ်းစားသင့်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

FUNCTIONS

❒ Functions

Functions တွေက C++ programming မှာ code တွေကို reusable အဖြစ် စုစည်းဖို့ သုံးပါတယ်။ Function တစ်ခုကို တစ်ခါသတ်မှတ်ပြီး အကြိမ်ကြိမ် ခေါ်သုံးနိုင်ပါတယ်။

1. အခြေခံ Function ဖွဲ့စည်းပုံ

```
#include <iostream>
using namespace std;

// Function declaration
int addNumbers(int a, int b);

int main() {
    int num1 = 5, num2 = 7;

    // Function call
    int result = addNumbers(num1, num2);

    cout << num1 << " + " << num2 << " = " << result << endl;
    return 0;
}

// Function definition
int addNumbers(int a, int b) {
    return a + b;
}
```

✓ ရှင်းလင်းချက်

1. Function Declaration - Function signature ကို အရင်ကြေငြာပါတယ် (return type, name, parameters)
2. Function Call - Program ထဲက နေရာမှာ function ကို ခေါ်သုံးပါတယ်
3. Function Definition - Function ရဲ့ အလုပ်လုပ်ပုံကို အသေးစိတ် ဖော်ပြပါတယ်

✓ ရလဒ်

```
5 + 7 = 12
```

2. Parameter အမျိုးအစားများ

Pass by Value

```
void square(int num) {
    num = num * num;
    cout << "Function အတွင်း:" << num << endl;
}

int main() {
    int x = 5;
    square(x);
    cout << "Function အပြင်:" << x << endl;
    return 0;
}
```

✓ ရှင်းလင်းချက်

- Original value ကို မပြောင်းလဲပါ (copy တစ်ခုကိုသာ အလုပ်လုပ်ပါတယ်)

✓ ရလဒ်

```
Function အတွင်း: 25
Function အပြင်: 5
```

Pass by Reference

```
void square(int &num) {
    num = num * num;
    cout << "Function အတွင်း:" << num << endl;
}

int main() {
    int x = 5;
    square(x);
    cout << "Function အပြင်:" << x << endl;
    return 0;
}
```

✓ ရှင်းလင်းချက်

- Original value ကို တိုက်ရိုက်ပြောင်းလဲပါတယ်

✓ ရလဒ်

```
Function အတွင်း: 25
```

```
Function အပြင်: 25
```

3. Default Parameters

```
#include <iostream>
using namespace std;

// Default parameter ပါတဲ့ function
void display(string message = "Hello, World!") {
    cout << message << endl;
}

int main() {
    display(); // Default parameter သုံးမယ်
    display("မင်္ဂလာပါ C++"); // Custom message ပို့မယ်
    return 0;
}
```

✓ ရလဒ်

```
Hello, World!
မင်္ဂလာပါ C++
```

4. Function Overloading

```
#include <iostream>
using namespace std;

// Function overloading ဥပမာများ
void print(int i) {
    cout << "Integer: " << i << endl;
}
```

```
void print(double f) {
    cout << "Float: " << f << endl;
}

void print(string s) {
    cout << "String: " << s << endl;
}

int main() {
    print(5);
    print(3.14);
    print("Hello C++");
    return 0;
}
```

✓ ရှင်းလင်းချက်

- တူညီတဲ့ function name နဲ့ parameter type အမျိုးမျိုးနဲ့ သတ်မှတ်နိုင်ပါတယ်

✓ ရလဒ်

```
Integer: 5
Float: 3.14
String: Hello C++
```

5. Recursive Functions

```
#include <iostream>
using namespace std;
```

```
// Factorial တွက်တဲ့ recursive function
int factorial(int n) {
    if (n <= 1)
        return 1;
    else
        return n * factorial(n - 1);
}

int main() {
    int num = 5;
    cout << num << "! = " << factorial(num) << endl;
    return 0;
}
```

✓ ရှင်းလင်းချက်

- Function ကိုယ်တိုင်ကိုပဲ ပြန်ခေါ်သုံးတဲ့နည်း

✓ ရလဒ်

```
5! = 120
```

6. Lambda Functions (C++11 နှင့်အထက်)

```
#include <iostream>
using namespace std;

int main() {
    // Lambda function ဥပမာ
    auto greet = []() {
        cout << "မင်္ဂလာပါ Lambda Functions!" << endl;
    };

    greet();
```

```
    // Parameter ပါတဲ့ lambda
    auto add = [](int a, int b) {
        return a + b;
    };

    cout << "10 + 15 = " << add(10, 15) << endl;

    return 0;
}
```

✓ Short Notes:

1. Functions တွေက code reuse လုပ်ဖို့အတွက် အရေးကြီးပါတယ်
2. Function တစ်ခုမှာ return type, name, parameters နဲ့ body ပါဝင်ပါတယ်
3. Pass by value vs pass by reference ကို သေချာနားလည်ဖို့လိုပါတယ်
4. Function overloading ကို သုံးပြီး တူညီတဲ့ name နဲ့ functions အမျိုးမျိုး ဖန်တီးနိုင်ပါတယ်
5. Recursive functions တွေက algorithm တချို့အတွက် အသုံးဝင်ပါတယ်
6. Lambda functions တွေက C++11 မှာ မိတ်ဆက်ပေးထားတဲ့ feature တစ်ခုဖြစ်ပါတယ်

Random

❒ Random Number Generation

C++11 နဲ့ အထက်မှာ <random> header ကို အသုံးပြုပြီး ပိုမိုကောင်းမွန်တဲ့ random number generation လုပ်နိုင်ပါတယ်။

1. Basic Random Number Generation

```
#include <iostream>
#include <random>  // Random number library

int main() {
    // Random device ကို initialize လုပ်ပါတယ်
    std::random_device rd;

    // Mersenne Twister engine ကို သုံးပါတယ်
    std::mt19937 gen(rd());

    // Distribution ကို သတ်မှတ်ပါတယ် (0 မှ 99 အထိ)
    std::uniform_int_distribution<> distrib(0, 99);

    // Random number 10 ခု ထုတ်ပါတယ်
    for (int i = 0; i < 10; ++i) {
        std::cout << distrib(gen) << ' ';
    }

    return 0;
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

2. Different Types of Distributions

```
#include <random>
#include <iostream>

int main() {
    std::random_device rd;
    std::mt19937 gen(rd());

    // Uniform integer distribution (1-6) - အန်စာတုံးလို
    std::uniform_int_distribution<> dice(1, 6);
    std::cout << "Dice roll: " << dice(gen) << '\n';

    // Uniform real distribution (0.0-1.0)
    std::uniform_real_distribution<> real_dist(0.0, 1.0);
    std::cout << "Real number: " << real_dist(gen) << '\n';

    // Normal distribution (mean=0, stddev=1)
    std::normal_distribution<> normal_dist(0.0, 1.0);
    std::cout << "Normal distribution: " << normal_dist(gen) << '\n';

    return 0;
}
```

Random String Generation (ကျပန်းစာသားများ ထုတ်ယူခြင်း)

1. Fixed Length Random String

```
#include <iostream>
#include <random>
#include <string>

std::string generate_random_string(int length) {
    std::random_device rd;
    std::mt19937 gen(rd());

    // Printable ASCII characters (32-126)
    std::uniform_int_distribution<> distrib(32, 126);
```

```
    std::string result;
    for (int i = 0; i < length; ++i) {
        result += static_cast<char>(distrib(gen));
    }

    return result;
}

int main() {
    std::cout << "Random string: " << generate_random_string(10) << '\n';
    return 0;
}
```

2. Random String with Specific Characters

```
#include <iostream>
#include <random>
#include <string>

std::string generate_random_alphanumeric(int length) {
    const std::string chars =
        "0123456789"
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        "abcdefghijklmnopqrstuvwxyz";

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(0, chars.size() - 1);

    std::string result;
    for (int i = 0; i < length; ++i) {
        result += chars[distrib(gen)];
    }

    return result;
}

int main() {
    std::cout << "Random alphanumeric: " << generate_random_alphanumeric(15)
<< '\n';
```

```
    return 0;
}
```

3. Random Hexadecimal String

```
#include <iostream>
#include <random>
#include <string>
#include <iomanip>
#include <sstream>

std::string generate_random_hex(int length) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(0, 255);

    std::stringstream ss;
    for (int i = 0; i < length; ++i) {
        int random_byte = distrib(gen);
        ss << std::hex << std::setw(2) << std::setfill('0') << random_byte;
    }

    return ss.str();
}

int main() {
    std::cout << "Random hex: " << generate_random_hex(16) << '\n';
    return 0;
}
```

✓ Seeding Random Generators

```
#include <iostream>
#include <random>
#include <chrono>

int main() {
    // Time-based seed
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count
();
    std::mt19937 gen(seed);
```

```
    std::uniform_int_distribution<> distrib(1, 100);
    std::cout << "Time-based random: " << distrib(gen) << '\n';

    // Fixed seed (for reproducible results)
    std::mt19937 gen2(12345);  // Fixed seed value
    std::cout << "Fixed seed random: " << distrib(gen2) << '\n';

    return 0;
}
```

Random Shuffle (ကျပန်းရောနှောခြင်း)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>

int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    std::random_device rd;
    std::mt19937 gen(rd());

    // Shuffle the vector
    std::shuffle(numbers.begin(), numbers.end(), gen);

    for (int num : numbers) {
        std::cout << num << ' ';
    }

    return 0;
}
```

✓ Practical Applications (လက်တွေ့အသုံးချနည်းများ)

1. Password Generation

```
std::string generate_password(int length) {
    const std::string chars =
        "0123456789"
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        "abcdefghijklmnopqrstuvwxyz"
        "!@#$%^&*()";

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(0, chars.size() - 1);

    std::string password;
    for (int i = 0; i < length; ++i) {
        password += chars[distrib(gen)];
    }

    return password;
}
```

2. Random ID Generation

```
std::string generate_uuid() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(0, 15);
    std::uniform_int_distribution<> distrib2(8, 11);

    const char* hex_chars = "0123456789abcdef";
    std::string uuid;

    for (int i = 0; i < 36; ++i) {
        if (i == 8 || i == 13 || i == 18 || i == 23) {
            uuid += '-';
        } else if (i == 14) {
            uuid += '4';
        } else if (i == 19) {
```

```
            uuid += hex_chars[distrib2(gen)];
        } else {
            uuid += hex_chars[distrib(gen)];
        }
    }

    return uuid;
}
```

C++ မှာ random number နဲ့ random string generation တွေကို ကောင်းစွာအသုံးပြုခြင်းဖြင့်သင့် application တွေကို ပိုမိုလုံခြုံစိတ်ချရစေပြီး random ဖြစ်မှုကို အာမခံနိုင်မှာဖြစ်ပါတယ်။

✓ လေ့ကျင့်ရန်

❒ Password Generator (Function-Based)

✓ Project Structure

```
PasswordGenerator/
└──── main.cpp    (Contains all functions and main program)
```

Complete Code Implementation

```
#include <iostream>
#include <string>
#include <random>
#include <algorithm>

// Function prototypes
void displayMenu();
int getPasswordLength();
std::string generateNumbersOnly(int length);
```

```
std::string generateLettersOnly(int length);
std::string generateMixedPassword(int length);
char getRandomChar(const std::string& charSet);

int main() {
    // Initialize random number generator
    std::random_device rd;
    std::mt19937 gen(rd());

    int choice;

    do {
        displayMenu();
        std::cin >> choice;

        // Clear input buffer
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

        int length;
        std::string password;

        switch(choice) {
            case 1:
                length = getPasswordLength();
                password = generateNumbersOnly(length);
                std::cout << "Generated Password: " << password << "\n\n";
                break;

            case 2:
                length = getPasswordLength();
                password = generateLettersOnly(length);
                std::cout << "Generated Password: " << password << "\n\n";
                break;

            case 3:
                length = getPasswordLength();
                password = generateMixedPassword(length);
                std::cout << "Generated Password: " << password << "\n\n";
                break;

            case 4:
                std::cout << "Exiting program...\n";
```

```
                break;

            default:
                std::cout << "Invalid choice! Please try again.\n\n";
        }

    } while(choice != 4);

    return 0;
}

// Display menu options
void displayMenu() {
    std::cout << "Password Generator Menu\n";
    std::cout << "1. Generate Numbers Only Password\n";
    std::cout << "2. Generate Letters Only Password\n";
    std::cout << "3. Generate Mixed Password (Numbers + Letters)\n";
    std::cout << "4. Exit\n";
    std::cout << "Enter your choice (1-4): ";
}

// Get valid password length from user
int getPasswordLength() {
    int length;
    while(true) {
        std::cout << "Enter password length (minimum 6): ";
        std::cin >> length;

        if(std::cin.fail() || length < 6) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'
);
            std::cout << "Invalid input. Minimum length is 6.\n";
        } else {
            break;
        }
    }
    return length;
}

// Generate random character from given character set
char getRandomChar(const std::string& charSet) {
```

```
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(0, charSet.size() - 1);
    return charSet[distrib(gen)];
}

// Generate password with numbers only
std::string generateNumbersOnly(int length) {
    const std::string numbers = "0123456789";
    std::string password;

    for(int i = 0; i < length; i++) {
        password += getRandomChar(numbers);
    }

    return password;
}

// Generate password with letters only
std::string generateLettersOnly(int length) {
    const std::string letters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR
STUVWXYZ";
    std::string password;

    for(int i = 0; i < length; i++) {
        password += getRandomChar(letters);
    }

    return password;
}

// Generate mixed password with numbers and letters
std::string generateMixedPassword(int length) {
    const std::string numbers = "0123456789";
    const std::string letters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR
STUVWXYZ";
    const std::string allChars = numbers + letters;

    std::string password;

    // Ensure at least one number and one letter
    password += getRandomChar(numbers);
```

```
    password += getRandomChar(letters);

    // Fill remaining characters
    for(int i = 2; i < length; i++) {
        password += getRandomChar(allChars);
    }

    // Shuffle the password
    std::random_device rd;
    std::mt19937 gen(rd());
    std::shuffle(password.begin(), password.end(), gen);

    return password;
}
```

✓ Testing Results

Test Case 1: Numbers Only Password

```
Password Generator Menu
1. Generate Numbers Only Password
2. Generate Letters Only Password
3. Generate Mixed Password (Numbers + Letters)
4. Exit
Enter your choice (1-4): 1
Enter password length (minimum 6): 8
Generated Password: 72940518
```

Test Case 2: Letters Only Password

```
Password Generator Menu
1. Generate Numbers Only Password
2. Generate Letters Only Password
3. Generate Mixed Password (Numbers + Letters)
4. Exit
Enter your choice (1-4): 2
```

```
Enter password length (minimum 6): 10
Generated Password: KjxYyFmQpR
```

Test Case 3: Mixed Password

```
Password Generator Menu
1. Generate Numbers Only Password
2. Generate Letters Only Password
3. Generate Mixed Password (Numbers + Letters)
4. Exit
Enter your choice (1-4): 3
Enter password length (minimum 6): 12
Generated Password: s7Dk9Lm4P2
```

Test Case 4: Invalid Length

```
Password Generator Menu
1. Generate Numbers Only Password
2. Generate Letters Only Password
3. Generate Mixed Password (Numbers + Letters)
4. Exit
Enter your choice (1-4): 1
Enter password length (minimum 6): 3
Invalid input. Minimum length is 6.
Enter password length (minimum 6): 6
Generated Password: 592847
```

Test Case 5: Invalid Menu Choice

```
Password Generator Menu
1. Generate Numbers Only Password
2. Generate Letters Only Password
3. Generate Mixed Password (Numbers + Letters)
```

```
4. Exit
Enter your choice (1-4): 5
Invalid choice! Please try again.
```

✓ ရှင်းလင်းချက်

1. Random Password Generation:
    - generateNumbersOnly(): နံပါတ်များသာပါသော password ထုတ်ပေးမယ်
    - generateLettersOnly(): စာလုံးများသာပါသော password ထုတ်ပေးမယ်
    - generateMixedPassword(): နံပါတ်နဲ့စာလုံးရောပါတဲ့ password ထုတ်ပေးမယ်

2. Input Validation:
    - Password အရှည်အနည်းဆုံး 6 ဖြစ်ရမယ်
    - Menu choice 1-4 ကြားဘဲ ရွေးချယ်နိုင်မယ်

3. Randomness:
    - C++ <random> library ကို အသုံးပြုထားတယ်
    - std::mt19937 Mersenne Twister engine ကို သုံးထားတယ်
    - Password များကို shuffle လုပ်ပေးထားတယ်

4. Function-Based Design:
    - Class တွေ file တွေမသုံးဘဲ function တွေနဲ့ဘဲ ရေးထားတယ်
    - ရိုးရှင်းပြီး နားလည်လွယ်တယ်

5. User-Friendly Interface:
    - ရှင်းလင်းတဲ့ menu system
    - အသုံးပြုရလွယ်ကူတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Date

❒ Date

C++ programming language မှာ date များကို ကိုင်တွယ်ဖို့ အခြေခံနည်းလမ်းများနှင့် library များအကြောင်း ရှင်းပြပေးပါမယ်။

1. <ctime> Library အခြေခံ

C++ မှာ ရက်စွဲများကို ကိုင်တွယ်ဖို့ အခြေခံ library က <ctime> (သို့) <time.h> ဖြစ်ပါတယ်။

```
#include <iostream>
#include <ctime>

int main() {
    // လက်ရှိ time ကို ရယူခြင်း
    time_t now = time(0);

    // time_t ကို string အဖြစ် ပြောင်းခြင်း
    char* dt = ctime(&now);

    std::cout << "လက်ရှိ ရက်စွဲနှင့် အချိန်: " << dt;

    return 0;
}
```

2. tm struct အသုံးပြုခြင်း

tm structure ကို အသုံးပြုပြီး date အစိတ်အပိုင်းများကို ခွဲထုတ်နိုင်ပါတယ်။

```
#include <iostream>
#include <ctime>
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
int main() {
    time_t now = time(0);
    tm *ltm = localtime(&now);

    std::cout << "နှစ်: " << 1900 + ltm->tm_year << std::endl;
    std::cout << "လ: " << 1 + ltm->tm_mon << std::endl;
    std::cout << "ရက်: " << ltm->tm_mday << std::endl;
    std::cout << "နေ့: " << ltm->tm_wday << std::endl;  // 0=တနင်္ဂနွေ, 1=တနင်္လာ, ...

    return 0;
}
```

3. C++20 <chrono> Library

C++20 မှာ ပိုမိုကောင်းမွန်တဲ့ <chrono> library ပါလာပါတယ်။

```
#include <iostream>
#include <chrono>
#include <format>

int main() {
    auto now = std::chrono::system_clock::now();

    // Format လုပ်ဖို့ (C++20)
    std::string date = std::format("{:%Y-%m-%d %H:%M:%S}", now);

    std::cout << "လက်ရှိ ရက်စွဲနှင့် အချိန်: " << date << std::endl;

    return 0;
}
```

4. Third-party Libraries

ပိုမိုအဆင့်မြင့် date လုပ်ဆောင်ချက်များအတွက်:

- Boost.Date_Time
- Howard Hinnant's date library

ဥပမာ Howard Hinnant's date library:

```
#include "date/date.h"
#include <iostream>

int main() {
    using namespace date;
    auto today = floor<days>(std::chrono::system_clock::now());
    std::cout << "ယနေ့ရက်စွဲ: " << today << '\n';


    return 0;
}
```

မှတ်ချက်: C++ မှာ date ကိုင်တွယ်မှုက အခြား programming language များလောက် မလွယ်ကူသေးပါဘူး။ ဒါကြောင့် third-party library များကို အသုံးပြုကြပါတယ်။

✓ လေ့ကျင့်ရန်

C++ Date Application - Weekday Calculator

ဒီ program မှာ သုံးစွဲသူက dd/mm/yyyy format နဲ့ ရက်စွဲကို ရိုက်ထည့်ရင် ဒီရက်စွဲဟာ Monday, Tuesday စသည်ဖြင့် ဘယ်နေ့ကျရောက်တယ်ဆိုတာ တွက်ပေးပါမယ်။

```
#include <iostream>
#include <ctime>
#include <sstream>
#include <iomanip>
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
using namespace std;

// Function to parse dd/mm/yyyy format and calculate weekday
string calculateWeekday(const string& inputDate) {
    int day, month, year;
    char sep; // separator (/)

    istringstream iss(inputDate);
    iss >> day >> sep >> month >> sep >> year;

    // Validate the date input
    if (iss.fail() || sep != '/' || day < 1 || day > 31 ||
        month < 1 || month > 12 || year < 1900) {
        return "မှားယွင်းသော ရက်စွဲ format! dd/mm/yyyy ဖြင့် ရိုက်ထည့်ပါ။";
    }

    // tm struct requires: year since 1900, month 0-11
    struct tm tm = {0};
    tm.tm_year = year - 1900;
    tm.tm_mon = month - 1;
    tm.tm_mday = day;

    // Normalize and calculate weekday
    mktime(&tm);

    // Weekday names in Burmese (optional)
    const char* weekdays[] = {
        "တနင်္ဂနွေ", "တနင်္လာ", "အင်္ဂါ", "ဗုဒ္ဓဟူး",
        "ကြာသပတေး", "သောကြာ", "စနေ"
    };

    return weekdays[tm.tm_wday];
}

int main() {
    string inputDate;

    cout << "ရက်စွဲကို dd/mm/yyyy format ဖြင့် ရိုက်ထည့်ပါ (ဥပမာ: 23/05/2023): ";
    cin >> inputDate;

    string weekday = calculateWeekday(inputDate);
```

```
    cout << "ဒီရက်စွဲဟာ " << weekday << " နေ့ ဖြစ်ပါတယ်။" << endl;

    return 0;
}
```

✓ Program အလုပ်လုပ်ပုံ

1. အချက်အလက်ရယူခြင်း
   - သုံးစွဲသူက dd/mm/yyyy format နဲ့ ရက်စွဲကို ရိုက်ထည့်ပါမယ် (ဥပမာ - 23/05/2023)
2. Data စစ်ဆေးခြင်း
   - ရိုက်ထည့်လိုက်တဲ့ format မှန်မမှန် စစ်ဆေးပါမယ်
   - ရက်လွန်၊ လကျော် ရိုက်ထည့်မိမထားဘူးဆိုတာ စစ်ဆေးပါမယ်
3. Weekday တွက်ချက်ခြင်း
   - C++ ရဲ့ <ctime> library ကို အသုံးပြုပြီး ရက်စွဲကို process လုပ်ပါမယ်
   - mktime() function က ရက်စွဲကို normalize လုပ်ပြီး weekday ကို တွက်ပေးပါမယ်
4. ရလဒ်ပြသခြင်း
   - တွက်ချက်ပြီးသား weekday ကို အင်္ဂလိပ်လို (သို့) မြန်မာလို ပြသပေးပါမယ်

✓ ဥပမာ Run ကြည့်ခြင်း

```
ရက်စွဲကို dd/mm/yyyy format ဖြင့် ရိုက်ထည့်ပါ (ဥပမာ: 23/05/2023): 23/05/2023
ဒီရက်စွဲဟာ တနင်္ဂနွေ နေ့ ဖြစ်ပါတယ်။


ရက်စွဲကို dd/mm/yyyy format ဖြင့် ရိုက်ထည့်ပါ (ဥပမာ: 23/05/2023): 31/12/2023
ဒီရက်စွဲဟာ တနင်္လာနွေ နေ့ ဖြစ်ပါတယ်။
```

✓ Short Notes:

1. ဒီ program က C++11 (သို့) နောက်ပိုင်း version တွေမှာ အလုပ်လုပ်ပါမယ်
2. မြန်မာလို နေ့အမည်တွေ ပြချင်ရင် weekdays array ကို ပြင်ဆင်နိုင်ပါတယ်
3. ရက်စွဲ validation ကို ပိုမိုတိကျစွာ လုပ်ဆောင်ချင်ရင် လတိုင်းရဲ့ရက်အရေအတွက်ကို ထည့်သွင်းစစ်ဆေးနိုင်ပါတယ်

String Manipulations

❒ String Manipulations

C++ programming မှာ string manipulation (စာသားများကို ကိုင်တွယ်ခြင်း) ဆိုတာ အလွန်အရေးကြီးတဲ့ အစိတ်အပိုင်းဖြစ်ပါတယ်။ C++ မှာ string များကို ကိုင်တွယ်ဖို့ အဓိက နည်းလမ်း ၂ မျိုးရှိပါတယ်:

1. C-style Strings (Character Arrays)

```
#include <iostream>
#include <cstring> // C-style string functions များအတွက်

int main() {
    char str1[20] = "Hello";
    char str2[20] = "World";

    // String length
    std::cout << "str1 length: " << strlen(str1) << std::endl;

    // String concatenation
    strcat(str1, " ");
    strcat(str1, str2);
    std::cout << "Concatenated: " << str1 << std::endl;

    // String copy
    char str3[20];
    strcpy(str3, str1);
    std::cout << "Copied string: " << str3 << std::endl;

    // String comparison
    if(strcmp(str1, str3) == 0) {
        std::cout << "Strings are equal" << std::endl;
    }

    return 0;
}
```

2. C++ std::string Class

```
#include <iostream>
#include <string> // C++ string class အတွက်

int main() {
    std::string s1 = "Hello";
    std::string s2 = "World";

    // String length
    std::cout << "s1 length: " << s1.length() << std::endl;

    // String concatenation
    std::string s3 = s1 + " " + s2;
    std::cout << "Concatenated: " << s3 << std::endl;

    // Substring
    std::string sub = s3.substr(6, 5); // position 6 မှစပြီး 5 characters
    std::cout << "Substring: " << sub << std::endl;

    // Find
    size_t pos = s3.find("World");
    if(pos != std::string::npos) {
        std::cout << "Found at position: " << pos << std::endl;
    }

    // Replace
    s3.replace(6, 5, "C++");
    std::cout << "After replace: " << s3 << std::endl;

    // Insert
    s3.insert(5, " Beautiful");
    std::cout << "After insert: " << s3 << std::endl;

    // Erase
    s3.erase(5, 10);
    std::cout << "After erase: " << s3 << std::endl;

    return 0;
}
```

✓ အရေးကြီးသော String Manipulation Operations များ:

1. String Concatenation (ပေါင်းစည်းခြင်း)
    - C-style: strcat()
    - C++: + operator သို့မဟုတ် append()
2. String Copy (ကူးယူခြင်း)
    - C-style: strcpy()
    - C++: = operator သို့မဟုတ် assign()
3. String Comparison (နှိုင်းယှဉ်ခြင်း)
    - C-style: strcmp()
    - C++: == operator သို့မဟုတ် compare()
4. String Length (အရှည်)
    - C-style: strlen()
    - C++: length() သို့မဟုတ် size()
5. Substring (စာကြောင်းခွဲ)
    - C++: substr()
6. Find and Replace (ရှာဖွေခြင်းနှင့် အစားထိုးခြင်း)
    - C++: find(), replace()

⇨ String Input Methods

```
#include <iostream>
#include <string>

int main() {
    std::string name;

    // Using cin (space မတိုင်ခင်အထိသာ ဖတ်မယ်)
    std::cout << "Enter your name (cin): ";
    std::cin >> name;
    std::cout << "Hello, " << name << std::endl;

    // Clear input buffer
    std::cin.ignore();

    // Using getline (တစ်ကြောင်းလုံးဖတ်မယ်)
    std::cout << "Enter your full name (getline): ";
    std::getline(std::cin, name);
    std::cout << "Hello, " << name << std::endl;

    return 0;
}
```

⇨ String Conversion

```
#include <iostream>
#include <string>
#include <sstream>

int main() {
    // Number to string
    int age = 25;
    std::string ageStr = std::to_string(age);
    std::cout << "Age as string: " << ageStr << std::endl;

    // String to number
    std::string numStr = "1234";
```

```
    int num = std::stoi(numStr);
    std::cout << "String as number: " << num * 2 << std::endl;

    // Using stringstream
    std::stringstream ss;
    ss << "The value is: " << 42;
    std::string result = ss.str();
    std::cout << result << std::endl;

    return 0;
}
```

မှတ်ချက်: C++ programming မှာ std::string class ကို အသုံးပြုတာက C-style strings ထက် ပိုမိုလွယ်ကူပြီး ဘေးကင်းပါတယ်။ ဒါကြောင့် ဖြစ်နိုင်ရင် std::string ကိုပဲ အသုံးပြုသင့်ပါတယ်။

✓ လေ့ကျင့်ရန်

❒ Data Conversion Application

ဒီ application မှာ အချက်အလက်အမျိုးအစား (data types) များကို တစ်ခုမှ တစ်ခုသို့ ပြောင်းလဲပေးမယ့် menu-driven program ကို ဖော်ပြပေးပါမယ်။

Program Code

```
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>

using namespace std;

// Function prototypes
```

```
void intToString();
void stringToInt();
void floatToString();
void stringToFloat();
void displayMenu();

int main() {
    int choice;

    do {
        displayMenu();
        cout << "လုပ်ဆောင်ချက် ရွေးချယ်ပါ (1-5): ";
        cin >> choice;

        // Clear input buffer
        cin.ignore();

        switch(choice) {
            case 1:
                intToString();
                break;
            case 2:
                stringToInt();
                break;
            case 3:
                floatToString();
                break;
            case 4:
                stringToFloat();
                break;
            case 5:
                cout << "Program မှ ထွက်ခွာပါမည်..." << endl;
                break;
            default:
                cout << "မှားယွင်းသော ရွေးချယ်မှု! 1 မှ 5 အထိ နံပါတ်များသာ ရွေးချယ်ပါ။" << endl
;
        }

        if(choice != 5) {
            cout << "\nEnter နှိပ်ပါ ဆက်လက်ဆောင်ရွက်ရန်...";
            cin.get();
```

```
        }

    } while(choice != 5);

    return 0;
}

void displayMenu() {
    system("cls"); // console ကို ရှင်းလင်းခြင်း (Windows အတွက်)
    // system("clear"); // Linux/macOS အတွက်

    cout << "====================================" << endl;
    cout << "    C++ Data Conversion Application" << endl;
    cout << "====================================" << endl;
    cout << "1. Integer မှ String သို့ ပြောင်းရန်" << endl;
    cout << "2. String မှ Integer သို့ ပြောင်းရန်" << endl;
    cout << "3. Float မှ String သို့ ပြောင်းရန်" << endl;
    cout << "4. String မှ Float သို့ ပြောင်းရန်" << endl;
    cout << "5. Program မှ ထွက်ခွာရန်" << endl;
    cout << "====================================" << endl;
}

void intToString() {
    int num;
    cout << "\nInteger ကို ရိုက်ထည့်ပါ: ";
    cin >> num;

    // Method 1: Using to_string()
    string str1 = to_string(num);

    // Method 2: Using stringstream
    stringstream ss;
    ss << num;
    string str2 = ss.str();

    cout << "to_string() အသုံးပြု၍: " << str1 << endl;
    cout << "stringstream အသုံးပြု၍: " << str2 << endl;
}

void stringToInt() {
```

```
    string str;
    cout << "\nNumber ပါသော String ကို ရိုက်ထည့်ပါ: ";
    getline(cin, str);

    try {
        // Method 1: Using stoi()
        int num1 = stoi(str);

        // Method 2: Using stringstream
        stringstream ss(str);
        int num2;
        ss >> num2;

        cout << "stoi() အသုံးပြု၍: " << num1 << endl;
        cout << "stringstream အသုံးပြု၍: " << num2 << endl;
    } catch(const invalid_argument& e) {
        cout << "မှားယွင်းသော input! Number ပါသော String သာ ရိုက်ထည့်ပါ။" << endl;
    } catch(const out_of_range& e) {
        cout << "အလွန်ကြီးမားသော နံပါတ်! Integer range ထဲတွင် မရှိပါ။" << endl;
    }
}

void floatToString() {
    float num;
    cout << "\nFloat ကို ရိုက်ထည့်ပါ: ";
    cin >> num;

    // Method 1: Using to_string()
    string str1 = to_string(num);

    // Method 2: Using stringstream with precision control
    stringstream ss;
    ss << fixed << setprecision(2) << num; // decimal places 2 နေရာအထိ
    string str2 = ss.str();

    cout << "to_string() အသုံးပြု၍: " << str1 << endl;
    cout << "stringstream အသုံးပြု၍ (decimal 2 နေရာ): " << str2 << endl;
}

void stringToFloat() {
```

```
    string str;
    cout << "\nFloat ပါသော String ကို ရိုက်ထည့်ပါ: ";
    getline(cin, str);

    try {
        // Method 1: Using stof()
        float num1 = stof(str);

        // Method 2: Using stringstream
        stringstream ss(str);
        float num2;
        ss >> num2;

        cout << fixed << setprecision(2);
        cout << "stof() အသုံးပြု၍: " << num1 << endl;
        cout << "stringstream အသုံးပြု၍: " << num2 << endl;
    } catch(const invalid_argument& e) {
        cout << "မှားယွင်းသော input! Float ပါသော String သာ ရိုက်ထည့်ပါ။" << endl;
    } catch(const out_of_range& e) {
        cout << "အလွန်ကြီးမားသော နံပါတ်! Float range ထဲတွင် မရှိပါ။" << endl;
    }
}
```

✓ Program အလုပ်လုပ်ပုံ

1. Menu Display
    - Program စတင်တိုင်း menu ၅ ခုကို ပြသပေးပါမယ်
    - သုံးစွဲသူက 1 မှ 5 အထိ နံပါတ်တစ်ခုကို ရွေးချယ်ရပါမယ်
2. Conversion Options (အချက်အလက်ပြောင်းလဲမှုများ)
    - Option 1: Integer ကို String အဖြစ်သို့ ပြောင်းလဲခြင်း
    - Option 2: String ကို Integer အဖြစ်သို့ ပြောင်းလဲခြင်း
    - Option 3: Float ကို String အဖြစ်သို့ ပြောင်းလဲခြင်း
    - Option 4: String ကို Float အဖြစ်သို့ ပြောင်းလဲခြင်း

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

- Option 5: Program မှ ထွက်ခွာခြင်း

3. Error Handling (အမှားများကို ကိုင်တွယ်ခြင်း)
   - မှားယွင်းသော input များအတွက် try-catch blocks များဖြင့် ကိုင်တွယ်ထားပါတယ်
   - Range errors နှင့် invalid arguments များကို ဖမ်းယူပြီး သင့်လျော်သော error message များ ပြသပေးပါတယ်

✓ ဥပမာ Run ကြည့်ခြင်း

```
==================================
    C++ Data Conversion Application
==================================
1. Integer မှ String သို့ ပြောင်းရန်
2. String မှ Integer သို့ ပြောင်းရန်
3. Float မှ String သို့ ပြောင်းရန်
4. String မှ Float သို့ ပြောင်းရန်
5. Program မှ ထွက်ခွာရန်
==================================
လုပ်ဆောင်ချက် ရွေးချယ်ပါ (1-5): 1

Integer ကို ရိုက်ထည့်ပါ: 42
to_string() အသုံးပြု၍: 42
stringstream အသုံးပြု၍: 42

Enter နှိပ်ပါ ဆက်လက်ဆောင်ရွက်ရန်...
```

```
==================================
    C++ Data Conversion Application
==================================
1. Integer မှ String သို့ ပြောင်းရန်
2. String မှ Integer သို့ ပြောင်းရန်
```

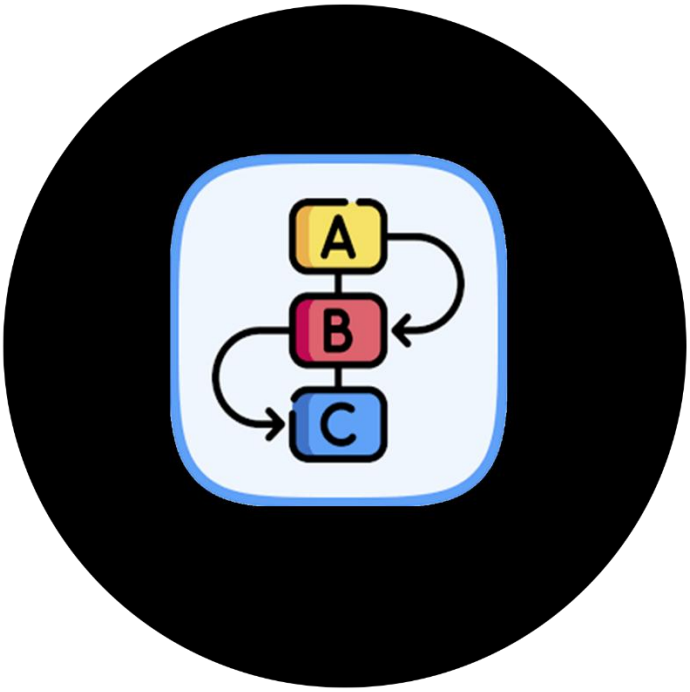
```
3. Float မှ String သို့ ပြောင်းရန်
4. String မှ Float သို့ ပြောင်းရန်
5. Program မှ ထွက်ခွာရန်
==================================
လုပ်ဆောင်ချက် ရွေးချယ်ပါ (1-5) : 3

Float ကို ရိုက်ထည့်ပါ: 3.14159
to_string() အသုံးပြု၍: 3.141590
stringstream အသုံးပြု၍ (decimal 2 နေရာ): 3.14

Enter နှိပ်ပါ ဆက်လက်ဆောင်ရွက်ရန်...
```

✓ Short Notes:

1. ဒီ program က C++11 သို့မဟုတ် နောက်ပိုင်း version များတွင် အလုပ်လုပ်ပါမယ်
2. Error handling များပါဝင်သောကြောင့် မှားယွင်းသော input များကို ကောင်းစွာ ကိုင်တွယ်နိုင်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

CLASS



❒ Class

Class တွေက C++ မှာ Object-Oriented Programming (OOP) ရဲ့အခြေခံဖြစ်ပြီး data နဲ့ functions တွေကို တစ်စုတစ်စည်းတည်း စုစည်းထားတဲ့ blueprint တစ်ခုဖြစ်ပါတယ်။

1. အခြေခံ Class ဖွဲ့စည်းပုံ

```
#include <iostream>
using namespace std;

// Car class ဖန်တီးခြင်း
class Car {
private:    // ဒီအတွင်းက data တွေကို class အပြင်ကနေ တိုက်ရိုက်မရနိုင်ဘူး
    string brand;
    string model;
    int year;

public:     // Public methods တွေကို class အပြင်ကနေ ခေါ်သုံးနိုင်တယ်
    // Constructor - object ဖန်တီးတဲ့အခါ အလိုအလျောက် run မယ်
    Car(string b, string m, int y) {
        brand = b;
        model = m;
        year = y;
    }

    // Method ကြေငြာခြင်း
    void displayInfo() {
        cout << "Brand: " << brand << endl;
        cout << "Model: " << model << endl;
        cout << "Year: " << year << endl;
    }
```

```
    // Setter method
    void setYear(int newYear) {
        if(newYear > 1886) {  // ပထမဆုံး ကားထုတ်တဲ့နှစ်
            year = newYear;
        }
    }

    // Getter method
    int getYear() {
        return year;
    }
};

int main() {
    // Car object ဖန်တီးခြင်း
    Car myCar("Toyota", "Corolla", 2020);

    // Object methods တွေကို ခေါ်သုံးခြင်း
    myCar.displayInfo();

    // Setter သုံးပြီး data ပြောင်းခြင်း
    myCar.setYear(2022);

    // Getter သုံးပြီး data ဖတ်ခြင်း
    cout << "\nUpdated Year: " << myCar.getYear() << endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. private members တွေက class အတွင်းကနေပဲ access လုပ်နိုင်ပါတယ်
2. public methods တွေက class အပြင်ကနေ ခေါ်သုံးနိုင်ပါတယ်
3. Constructor က object ဖန်တီးတဲ့အခါ initialize လုပ်ဖို့သုံးပါတယ်
4. Getter/Setter methods တွေက data encapsulation အတွက် အရေးကြီးပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ ရလဒ်

```
Brand: Toyota
Model: Corolla
Year: 2020

Updated Year: 2022
```

2. Constructor အမျိုးအစားများ

```
#include <iostream>
using namespace std;

class Student {
    string name;
    int age;
    double gpa;

public:
    // 1. Default Constructor
    Student() {
        name = "Unknown";
        age = 0;
        gpa = 0.0;
    }

    // 2. Parameterized Constructor
    Student(string n, int a) {
        name = n;
        age = a;
        gpa = 0.0;
    }

    // 3. Parameterized Constructor (all fields)
    Student(string n, int a, double g) {
        name = n;
        age = a;
        gpa = g;
    }
```

```
    void display() {
        cout << "Name: " << name << ", Age: " << age << ", GPA: " << gpa << e
ndl;
    }
};

int main() {
    // Constructor အမျိုးမျိုးသုံးပြီး object တွေဖန်တီးခြင်း
    Student s1;  // Default constructor
    Student s2("Aung Aung", 20);  // 2-parameter constructor
    Student s3("Hla Hla", 21, 3.5);  // 3-parameter constructor

    s1.display();
    s2.display();
    s3.display();

    return 0;
}
```

✓ ရလဒ်

```
Name: Unknown, Age: 0, GPA: 0
Name: Aung Aung, Age: 20, GPA: 0
Name: Hla Hla, Age: 21, GPA: 3.5
```

3. Inheritance (အမွေဆက်ခံခြင်း)

```
#include <iostream>
using namespace std;

// Base class (Parent class)
class Animal {
protected:  // Child class တွေက access လုပ်နိုင်တယ်
    string name;

public:
    Animal(string n) : name(n) {}
```

```
    void eat() {
        cout << name << " is eating." << endl;
    }
};
```

```
// Derived class (Child class)
class Dog : public Animal {
private:
    string breed;

public:
    Dog(string n, string b) : Animal(n), breed(b) {}

    void bark() {
        cout << name << " (" << breed << ") is barking: Woof! Woof!" << endl;
    }
};

int main() {
    Dog myDog("Bobby", "Golden Retriever");
    myDog.eat();  // Parent class method
    myDog.bark(); // Child class method

    return 0;
}
```

✓ ရှင်းလင်းချက်

- Animal class က Dog class ရဲ့ parent class ဖြစ်ပါတယ်
- protected members တွေက child classes တွေကနေ access လုပ်နိုင်ပါတယ်
- Dog class က Animal class ရဲ့ features တွေကို inherit လုပ်ထားပါတယ်

✓ ရလဒ်

```
Bobby is eating.
Bobby (Golden Retriever) is barking: Woof! Woof!
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

4. Polymorphism (ပုံစံအမျိုးမျိုးပြောင်းလဲနိုင်ခြင်း)

```
#include <iostream>
using namespace std;

class Shape {
public:
    // Virtual function - child classes တွေက override လုပ်နိုင်တယ်
    virtual void draw() {
        cout << "Drawing a shape" << endl;
    }
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing a circle" << endl;
    }
};

class Rectangle : public Shape {
public:
    void draw() override {
        cout << "Drawing a rectangle" << endl;
    }
};

int main() {
    Shape* shape1 = new Circle();
    Shape* shape2 = new Rectangle();

    shape1->draw();  // Circle ရဲ့ draw() method ကို call မယ်
    shape2->draw();  // Rectangle ရဲ့ draw() method ကို call မယ်

    delete shape1;
    delete shape2;

    return 0;
}
```

✓ ရလဒ်

```
Drawing a circle
Drawing a rectangle5. Static Class Members
```

```
#include <iostream>
using namespace std;

class Counter {
    static int count;  // Static member variable

public:
    Counter() {
        count++;  // Object တစ်ခုဆောက်တိုင်း count တိုးမယ်
    }

    static int getCount() {  // Static method
        return count;
    }
};

// Static variable initialization
int Counter::count = 0;

int main() {
    cout << "Initial count: " << Counter::getCount() << endl;

    Counter c1, c2, c3;

    cout << "After creating 3 objects: " << Counter::getCount() << endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

- Static members တွေက class level မှာရှိပြီး object တိုင်းအတွက် shared ဖြစ်ပါတယ်
- Static methods တွေက object instance မလိုဘဲ ခေါ်သုံးနိုင်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ ရလဒ်

```
Initial count: 0
After creating 3 objects: 3
```

✓ Short Notes:

1. Class တွေက data နဲ့ functions တွေကို စုစည်းထားတဲ့ blueprint တွေဖြစ်ပါတယ်
2. Encapsulation က data တွေကို protect လုပ်ဖို့သုံးပါတယ်
3. Inheritance က code reuse လုပ်ဖို့အတွက် အသုံးဝင်ပါတယ်
4. Polymorphism က interface တစ်ခုတည်းနဲ့ different implementations တွေကို သုံးနိုင်ပါတယ်
5. Constructors တွေက object initialization အတွက် အရေးကြီးပါတယ်
6. Static members တွေက class level မှာ shared ဖြစ်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

METHODS

❒ Methods

Methods (သို့မဟုတ် member functions) တွေက class အတွင်းမှာ သတ်မှတ်ထားတဲ့ functions တွေဖြစ်ပြီး object တွေရဲ့ behavior တွေကို ဖော်ပြပါတယ်။

1. Basic Method Definition and Usage

```
#include <iostream>
using namespace std;

class Calculator {
public:
    // Method ကြေငြာခြင်း
    int add(int a, int b) {
        return a + b;
    }

    // Parameter မပါတဲ့ method
    void greet() {
        cout << "Hello from Calculator class!" << endl;
    }
};

int main() {
    Calculator calc; // Object ဖန်တီးခြင်း

    // Methods တွေကို ခေါ်သုံးခြင်း
    int result = calc.add(5, 3);
    cout << "5 + 3 = " << result << endl;

    calc.greet();

    return 0;
```

```
}
```

✓ ရှင်းလင်းချက်

1. add method က parameter နှစ်ခုလက်ခံပြီး ပေါင်းလဒ်ကို return ပြန်ပေးပါတယ်
2. greet method က parameter မလိုဘဲ message တစ်ခုကို ရိုက်ထုတ်ပါတယ်
3. Object ကနေ dot operator (.) သုံးပြီး methods တွေကို ခေါ်သုံးပါတယ်

✓ ရလဒ်

```
5 + 3 = 8
Hello from Calculator class!
```

2. Getter and Setter Methods

```
#include <iostream>
using namespace std;

class BankAccount {
private:
    double balance;

public:
    // Setter method
    void setBalance(double newBalance) {
        if(newBalance >= 0) {
            balance = newBalance;
        } else {
            cout << "Error: Balance cannot be negative" << endl;
        }
    }

    // Getter method
    double getBalance() {
        return balance;
    }
```

```
};

int main() {
    BankAccount account;

    account.setBalance(1000.50);
    cout << "Current balance: " << account.getBalance() << endl;

    account.setBalance(-500); // Error message ပြမယ်
    cout << "Balance after invalid update: " << account.getBalance() << endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. balance variable က private ဖြစ်တဲ့အတွက် ပြင်ပကနေ တိုက်ရိုက်မဝင်ရောက်နိုင်ပါ
2. setBalance method ကနေ validation လုပ်ပြီး balance ကို update လုပ်ပါတယ်
3. getBalance method က current balance ကို return ပြန်ပေးပါတယ်

✓ ရလဒ်

```
Current balance: 1000.5
Error: Balance cannot be negative
Balance after invalid update: 1000.5
```

3. Const Methods (မပြောင်းလဲနိုင်သော methods)

```
#include <iostream>
using namespace std;

class Student {
    string name;
```

```
    int age;

public:
    Student(string n, int a) : name(n), age(a) {}

    // Const method - object data တွေကို မပြောင်းလဲဘူးဆိုတာ အာမခံချက်
    void displayInfo() const {
        cout << "Name: " << name << ", Age: " << age << endl;
        // name = "Changed"; // Error: const method ထဲမှာ modify လုပ်လို့မရ
    }

    // Non-const method
    void increaseAge() {
        age++;
    }
};

int main() {
    const Student s1("Alice", 20);
    // s1.increaseAge(); // Error: const object ကနေ non-const method ကို ခေါ်လို့မရ
    s1.displayInfo(); // OK: const method ကို const object ကနေ ခေါ်လို့ရ

    Student s2("Bob", 22);
    s2.increaseAge(); // OK
    s2.displayInfo(); // OK

    return 0;
}
```

4. Method Overloading (တူညီတဲ့ method name နဲ့ parameter အမျိုးမျိုး)

```
#include <iostream>
using namespace std;

class MathOperations {
public:
    // ကိန်းပြည့်နှစ်လုံးပေါင်းခြင်း
    int add(int a, int b) {
        return a + b;
```

```
    }

    // ဒဿမနှစ်လုံးပေါင်းခြင်း
    double add(double a, double b) {
        return a + b;
    }

    // ကိန်းသုံးလုံးပေါင်းခြင်း
    int add(int a, int b, int c) {
        return a + b + c;
    }
};

int main() {
    MathOperations math;

    cout << "5 + 3 = " << math.add(5, 3) << endl;
    cout << "4.2 + 1.8 = " << math.add(4.2, 1.8) << endl;
    cout << "10 + 20 + 30 = " << math.add(10, 20, 30) << endl;

    return 0;
}
```

✓ ရလဒ်

```
5 + 3 = 8
4.2 + 1.8 = 6
10 + 20 + 30 = 60
```

5. Static Methods (Class-level methods)

```
#include <iostream>
using namespace std;

class Counter {
    static int count; // Static member variable

public:
```

```
    Counter() {
        count++;
    }

    // Static method
    static int getCount() {
        return count;
    }
};
```

```
// Static variable initialization
int Counter::count = 0;

int main() {
    cout << "Initial count: " << Counter::getCount() << endl;

    Counter c1, c2, c3;

    cout << "After creating 3 objects: " << Counter::getCount() << endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. Static methods တွေက object-specific မဟုတ်ဘဲ class-level ဖြစ်ပါတယ်
2. Counter::getCount() လို့ class name ကနေ တိုက်ရိုက်ခေါ်သုံးနိုင်ပါတယ်
3. Static methods တွေက static member variables တွေကိုပဲ access လုပ်နိုင်ပါတယ်

✓ ရလဒ်

```
Initial count: 0
After creating 3 objects: 3
```

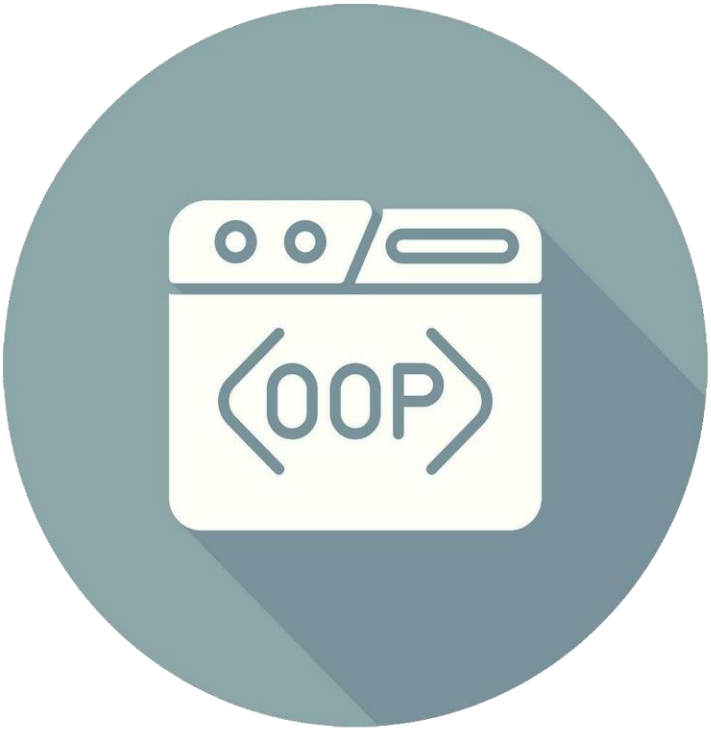

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ Short Notes:

1. Methods တွေက class ရဲ့ behavior တွေကို ဖော်ပြပါတယ်
2. Access specifiers (public, private, protected) တွေက method တွေရဲ့ visibility ကို သတ်မှတ်ပေးပါတယ်
3. const methods တွေက object data တွေကို မပြောင်းလဲဘူးဆိုတာ အာမခံပါတယ်
4. Method overloading ကို သုံးပြီး တူညီတဲ့ name နဲ့ parameter အမျိုးမျိုးနဲ့ methods တွေ ဖန်တီးနိုင်ပါတယ်
5. Static methods တွေက object instance မလိုဘဲ ခေါ်သုံးနိုင်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

OOP



❒ OOP

C++ မှာ Object-Oriented Programming (OOP) ဆိုတာက programming paradigm တစ်ခုဖြစ်ပြီး program တွေကို object တွေအဖြစ် စီမံခန့်ခွဲရေးသားနည်းဖြစ်ပါတယ်။

✓ OOP ရဲ့အခြေခံ အယူအဆများ

1. Class နှင့် Object

- Class ဆိုတာက object တွေရဲ့ blueprint (ဒီဇိုင်း) ဖြစ်ပါတယ်
- Object ဆိုတာက class ကနေ ဖန်တီးထားတဲ့ instance တစ်ခုဖြစ်ပါတယ်

```
class Car {  // Class
public:
    string brand;
    void honk() {
        cout << "Beep beep!" << endl;
    }
};

int main() {
    Car myCar;  // Object
    myCar.brand = "Toyota";
    myCar.honk();
    return 0;
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

2. Encapsulation (ထုပ်ပိုးခြင်း)

- Data နဲ့ function တွေကို class တစ်ခုထဲမှာ စုစည်းပြီး၊ data တွေကို direct access မရအောင် ကာကွယ်ပေးတာဖြစ်ပါတယ်
- private, public, protected access modifiers တွေကို သုံးပါတယ်

```
class BankAccount {
private:
    double balance;  // encapsulated data

public:
    void deposit(double amount) {
        if(amount > 0) balance += amount;
    }

    double getBalance() {
        return balance;
    }
};
```

3. Inheritance (အမွေဆက်ခံခြင်း)

- Existing class တစ်ခုကနေ new class တစ်ခုကို create လုပ်တာဖြစ်ပါတယ်
- Code reuse လုပ်နိုင်ပြီး hierarchical relationship တွေကို ဖော်ပြနိုင်ပါတယ်

```
class Animal {  // Base class
public:
    void eat() {
        cout << "Eating..." << endl;
    }
};

class Dog : public Animal {  // Derived class
```

```
public:
    void bark() {
        cout << "Barking..." << endl;
    }
};

int main() {
    Dog myDog;
    myDog.eat();  // Inherited from Animal
    myDog.bark();
    return 0;
}
```

4. Polymorphism (ပုံစံ အမျိုးမျိုးပြောင်းလဲနိုင်ခြင်း)

- Single action ကို different ways နဲ့ perform လုပ်နိုင်တဲ့ ability ဖြစ်ပါတယ်
- Function overloading နဲ့ function overriding တွေကို သုံးပါတယ်

```
class Shape {
public:
    virtual void draw() {  // Virtual function
        cout << "Drawing a shape" << endl;
    }
};

class Circle : public Shape {
public:
    void draw() override {  // Function overriding
        cout << "Drawing a circle" << endl;
    }
};

int main() {
    Shape* shape = new Circle();
    shape->draw();  // Calls Circle's draw()
    delete shape;
    return 0;
}
```

ဆက်လက်ပြီး OOP ရဲ့ concept တွေကို အသေးစိတ်လေ့လာကြရအောင်။ လေ့ကျင့်ခန်းတွေကိုလည်း လက်တွေ့လိုက်လုပ်ကြည့်ဖို့ တိုက်တွန်းပါတယ်။ ဆက်ပြီး ဆွေးနွေးပေးမည် OOP concept ခေါင်းစဉ်တွေက တော့ -

- Constructor
- Destructor
- Encapsulations
- Inheritance
- Polymorphism
- Abstractions

တွေပဲ ဖြစ်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ Constructor

Constructor ဆိုတာက class ကနေ object တစ်ခုကို ဖန်တီးတဲ့အခါ အလိုအလျောက် ခေါ်တဲ့ special member function တစ်ခုဖြစ်ပါတယ်။

✓ Constructor ရဲ့အရေးကြီးတဲ့ အချက်များ:

1. Constructor က object ဖန်တီးတိုင်း အလိုအလျောက် ခေါ်ပါတယ်
2. Constructor ရဲ့နာမည်က class နာမည်နဲ့ တူပါတယ်
3. Constructor က return type မရှိပါ (void တောင်မပါဘူး)
4. Constructor တွေကို overload လုပ်နိုင်ပါတယ်

✓ Constructor အမျိုးအစားများ:

1. Default Constructor (အလိုအလျောက် တည်ဆောက်သူ)

```
class Student {
public:
    Student() {  // Default Constructor
        cout << "Default Constructor ခေါ်ပါတယ်" << endl;
    }
};
```

```
int main() {
    Student s1;  // Default Constructor အလိုအလျောက်ခေါ်
    return 0;
}
```

2. Parameterized Constructor (အချက်ပြ တည်ဆောက်သူ)

```
class Student {
public:
    int id;
    Student(int x) {  // Parameterized Constructor
        id = x;
        cout << "Parameterized Constructor ခေါ်ပါတယ်, ID: " << id << endl;
    }
};

int main() {
    Student s2(101);  // Parameterized Constructor ခေါ်
    return 0;
}
```

3. Copy Constructor (ကူးယူ တည်ဆောက်သူ)

```
class Student {
public:
    int id;
    Student(int x) { id = x; }
    Student(Student &s) {  // Copy Constructor
        id = s.id;
        cout << "Copy Constructor ခေါ်ပါတယ်, ID: " << id << endl;
    }
};

int main() {
    Student s3(102);
    Student s4(s3);  // Copy Constructor ခေါ်
    return 0;
```

```
}
```

❒ Destructor

Destructor ဆိုတာက object ကို ဖျက်သိမ်းတဲ့အခါ အလိုအလျောက် ခေါ်တဲ့ special member function တစ်ခုဖြစ်ပါတယ်။

✓ Destructor ရဲ့အရေးကြီးတဲ့ အချက်များ:

1. Destructor က object ဖျက်သိမ်းတိုင်း အလိုအလျောက် ခေါ်ပါတယ်
2. Destructor ရဲ့နာမည်က class နာမည်နဲ့ တူပြီး ~ သင်္ကေတ ပါပါတယ်
3. Destructor က return type မရှိပါ (void တောင်မပါဘူး)
4. Destructor ကို parameter မယူပါ (overload လုပ်လို့မရပါ)

✓ Destructor ဥပမာ:

```
class Student {
public:
    int *marks;

    Student() {  // Constructor
        marks = new int[5];
        cout << "Memory allocate လုပ်ပါတယ်" << endl;
    }

    ~Student() {  // Destructor
        delete[] marks;
        cout << "Memory free လုပ်ပါတယ်" << endl;
    }
};
```

```
int main() {
    Student s5;
    // main function ပြီးတဲ့အခါ Destructor အလိုအလျောက်ခေါ်
    return 0;
}
```

✓ Constructor နှင့် Destructor အသုံးပြုရမယ့် အခြေအနေများ

1. Constructor ကို object initialize လုပ်တဲ့အခါ (ဥပမာ - memory allocation, variable တွေကို initial value ထည့်တဲ့အခါ)
2. Destructor ကို object ဖျက်သိမ်းတဲ့အခါ (ဥပမာ - allocated memory တွေကို free လုပ်တဲ့အခါ, file တွေကို close လုပ်တဲ့အခါ)

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ Encapsulations

Encapsulation (အထုပ်ပိုးခြင်း) ဆိုတာက OOP ရဲ့အရေးကြီးတဲ့ feature တစ်ခုဖြစ်ပြီး data နဲ့ functions တွေကို class တစ်ခုထဲမှာ စုစည်းကာ data တွေကို direct access မရအောင် ကာကွယ်ပေးတဲ့ နည်းလမ်းဖြစ်ပါတယ်။

✓ Encapsulation ရဲ့အဓိက ရည်ရွယ်ချက်များ

1. Data hiding - Class အတွင်းက data တွေကို အပြင်ကနေ တိုက်ရိုက် မပြင်နိုင်အောင် လုပ်ပါတယ်
2. Controlled access - Data တွေကို getter/setter methods တွေကနေဘဲ ထိန်းချုပ်ပြီး access လုပ်ပါတယ်
3. Security - Data တွေရဲ့ integrity ကို ကာကွယ်ပါတယ်
4. Implementation hiding - Internal implementation ကို ဖုံးကွယ်ထားပါတယ်

✓ Encapsulation ကို C++ မှာ အကောင်အထည်ဖော်နည်း

Access Modifiers (အသုံးပြုခွင့် သတ်မှတ်ချက်များ)

1. private - Class အတွင်းကသာ access လုပ်နိုင်ပါတယ်
2. public - ဘယ်နေရာကမဆို access လုပ်နိုင်ပါတယ်
3. protected - Inherited class တွေကနေ access လုပ်နိုင်ပါတယ်

```
class BankAccount {
private:    // Encapsulated data (အပြင်ကနေ တိုက်ရိုက်မရ)
    double balance;
    string accountNumber;

public:     // Public interface (အပြင်ကနေ access လုပ်နိုင်တဲ့ methods)
    // Setter method (data တွေကို စစ်ဆေးပြီး assign လုပ်ပေး)
    void setBalance(double amount) {
        if(amount >= 0) {
            balance = amount;
        } else {
            cout << "Invalid amount!" << endl;
        }
    }

    // Getter method (data တွေကို ပြန်ထုတ်ပေး)
    double getBalance() {
        return balance;
    }

    void deposit(double amount) {
        if(amount > 0) {
            balance += amount;
            cout << amount << " ထည့်သွင်းပြီးပါပြီ။ လက်ကျန်ငွေ: " << balance << endl;
        }
    }

    void withdraw(double amount) {
        if(amount > 0 && amount <= balance) {
            balance -= amount;
            cout << amount << " ထုတ်ယူပြီးပါပြီ။ လက်ကျန်ငွေ: " << balance << endl;
        } else {
            cout << "Invalid transaction!" << endl;
        }
    }
};
```

```
int main() {
    BankAccount account;

    // account.balance = 1000;  // Error! private member ကို တိုက်ရိုက် access မရ

    account.setBalance(5000);    // Setter method ကို သုံး
    cout << "လက်ရှိလက်ကျန်ငွေ:" << account.getBalance() << endl;  // Getter metho
d ကို သုံး

    account.deposit(2000);
    account.withdraw(1000);

    return 0;
}
```

✓ Encapsulation ရဲ့အကျိုးကျေးဇူးများ

1. Data Security - Data တွေကို unauthorized access ကနေ ကာကွယ်ပေးပါတယ်
2. Flexibility - Internal implementation ကို ပြောင်းလဲရင် class ကို သုံးထားတဲ့ code တွေကို ပြန်ပြင်စရာမလိုပါ
3. Reusability - Well-encapsulated class တွေကို အခြား program တွေမှာ ပြန်သုံးနိုင်ပါတယ်
4. Maintainability - Code တွေကို ပြုပြင်ရတာ ပိုလွယ်ကူပါတယ်

✓ Real-world Example - လူတစ်ဦး၏ အချက်အလက်

```
class Person {
private:
    string name;
    int age;
    string nrc;

public:
    // Setter methods
    void setName(string n) {
```

```
        if(!n.empty()) {
            name = n;
        }
    }

    void setAge(int a) {
        if(a > 0 && a < 120) {
            age = a;
        }
    }

    // Getter methods
    string getName() {
        return name;
    }

    int getAge() {
        return age;
    }

    string getNrcMasked() {
        if(nrc.length() >= 6) {
            return "*****" + nrc.substr(nrc.length()-4);
        }
        return "Invalid NRC";
    }
};

int main() {
    Person person;
    person.setName("မောင်မောင်");
    person.setAge(25);

    cout << "အမည်:" << person.getName() << endl;
    cout << "အသက်:" << person.getAge() << endl;

    return 0;
}
```

❒ Inheritance

Inheritance (အမွေဆက်ခံခြင်း) ဆိုတာက OOP ရဲ့အရေးကြီးတဲ့ feature တစ်ခုဖြစ်ပြီး existing class တစ်ခုကနေ new class တစ်ခုကို create လုပ်တဲ့နည်းလမ်းဖြစ်ပါတယ်။

✓ Inheritance ရဲ့အဓိက ရည်ရွယ်ချက်များ

1. Code Reusability - Existing code တွေကို ပြန်သုံးနိုင်ပါတယ်
2. Method Overriding - Base class က methods တွေကို modify လုပ်နိုင်ပါတယ်
3. Hierarchical Classification - Real-world relationships တွေကို ကောင်းစွာ represent လုပ်နိုင်ပါတယ်

✓ Inheritance အမျိုးအစားများ

1. Single Inheritance (တစ်ဆင့်အမွေဆက်ခံခြင်း)

```
class Animal {  // Base class (မိဘ class)
public:
    void eat() {
        cout << "စားနေပါတယ်..." << endl;
    }
};

class Dog : public Animal {  // Derived class (သားသမီး class)
public:
    void bark() {
        cout << "ဟောင်နေပါတယ်..." << endl;
    }
};

int main() {
```

```
    Dog myDog;
    myDog.eat();  // Base class ကနေ inherit လုပ်ထားတဲ့ method
    myDog.bark(); // Derived class က method
    return 0;
}
```

2. Multiple Inheritance

```
class Father {
public:
    void swim() {
        cout << "ဖခင် ရေကူးနိုင်တယ်" << endl;
    }
};

class Mother {
public:
    void cook() {
        cout << "မိခင် ချက်ပြုတ်နိုင်တယ်" << endl;
    }
};

class Child : public Father, public Mother {  // နှစ်လုံးကနေ inherit
public:
    void play() {
        cout << "ကလေး ကစားနိုင်တယ်" << endl;
    }
};

int main() {
    Child child;
    child.swim();  // Father ကနေ inherit
    child.cook();  // Mother ကနေ inherit
    child.play();  // Child က method
    return 0;
}
```

3. Multilevel Inheritance (အဆင့်ဆင့်အမွေဆက်ခံခြင်း)

```
class Grandparent {
public:
    void tellStory() {
        cout << "ဘိုးဘွား ပုံပြင်ပြောနေတယ်" << endl;
    }
};

class Parent : public Grandparent {
public:
    void work() {
        cout << "မိဘ အလုပ်လုပ်နေတယ်" << endl;
    }
};

class Child : public Parent {
public:
    void study() {
        cout << "ကလေး စာကျက်နေတယ်" << endl;
    }
};

int main() {
    Child child;
    child.tellStory();  // Grandparent ကနေ inherit
    child.work();       // Parent ကနေ inherit
    child.study();      // Child က method
    return 0;
}
```

4. Hierarchical Inheritance (အဆင့်လိုက်အမွေဆက်ခံခြင်း)

```
class Vehicle {
public:
    void move() {
        cout << "ယာဉ် ရွေ့လျားနေတယ်" << endl;
    }
};

class Car : public Vehicle {
```

```
public:
    void drive() {
        cout << "ကား မောင်းနေတယ်" << endl;
    }
};

class Bike : public Vehicle {
public:
    void ride() {
        cout << "စက်ဘီး စီးနေတယ်" << endl;
    }
};

int main() {
    Car car;
    Bike bike;

    car.move();  // Vehicle ကနေ inherit
    car.drive();

    bike.move();  // Vehicle ကနေ inherit
    bike.ride();

    return 0;
}
```

✓ Inheritance Access Specifiers

1. Public Inheritance

```
class Base {
public:
    int x;
protected:
    int y;
private:
    int z;
};
```

```
class Derived : public Base {
    // x public အဖြစ် ဆက်ရှိ
    // y protected အဖြစ် ဆက်ရှိ
    // z inaccessible (မရနိုင်)
};
```

2. Protected Inheritance

```
class Derived : protected Base {
    // x protected အဖြစ် ပြောင်း
    // y protected အဖြစ် ဆက်ရှိ
    // z inaccessible (မရနိုင်)
};
```

3. Private Inheritance

```
class Derived : private Base {
    // x private အဖြစ် ပြောင်း
    // y private အဖြစ် ပြောင်း
    // z inaccessible (မရနိုင်)
};
```

✓ Function Overriding (လုပ်ဆောင်ချက်အသစ်ပြန်သတ်မှတ်ခြင်း)

```
class Animal {
public:
    virtual void sound() {  // Virtual function
        cout << "တိရစ္ဆာန်အသံ" << endl;
    }
};

class Cat : public Animal {
```

```
public:
    void sound() override {  // Function overriding
        cout << "ပြောင်~" << endl;
    }
};

class Dog : public Animal {
public:
    void sound() override {  // Function overriding
        cout << "ဝါ~" << endl;
    }
};

int main() {
    Animal* animal1 = new Cat();
    Animal* animal2 = new Dog();

    animal1->sound();  // Cat က sound() ကို ခေါ်
    animal2->sound();  // Dog က sound() ကို ခေါ်

    delete animal1;
    delete animal2;
    return 0;
}
```

❒ POLYMORPHISM

Polymorphism (အမျိုးမျိုးပြောင်းလဲနိုင်ခြင်း) ဆိုတာက OOP ရဲ့အရေးကြီးဆုံး concept တစ်ခုဖြစ်ပြီး "တစ်ခုတည်းသော interface ကို အမျိုးမျိုးသော implementation တွေနဲ့ အသုံးပြုနိုင်ခြင်း" လို့ အဓိပ္ပာယ်ဖွင့်ဆိုနိုင်ပါတယ်။

✓ Polymorphism ရဲ့အဓိက အမျိုးအစားများ

1. Compile-time Polymorphism (Static Polymorphism)

- Function Overloading နဲ့ Operator Overloading တို့ပါဝင်ပါတယ်
- Program ကို compile လုပ်တဲ့အချိန်မှာ ဘယ် function ကို ခေါ်မယ်ဆိုတာ ဆုံးဖြတ်ပါတယ်

✓ Function Overloading ဥပမာ:

```
class Calculator {
public:
    // အမည်တူ functions တွေကို parameters အမျိုးမျိုးနဲ့ သတ်မှတ်ခြင်း
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    string add(string a, string b) {
        return a + b;
    }
};

int main() {
```

```
    Calculator calc;
    cout << calc.add(5, 3) << endl;        // 8
    cout << calc.add(2.5, 3.7) << endl;    // 6.2
    cout << calc.add("Hello", "World") << endl; // HelloWorld
    return 0;
}
```

2. Run-time Polymorphism (Dynamic Polymorphism)

- Function Overriding နဲ့ Virtual Functions တို့ပါဝင်ပါတယ်
- Program run နေစဉ်မှာ ဘယ် function ကို ခေါ်မယ်ဆိုတာ ဆုံးဖြတ်ပါတယ်

✓ Virtual Function နဲ့ Function Overriding ဥပမာ:

```
class Animal {
public:
    virtual void makeSound() {  // Virtual function
        cout << "တိရစ္ဆာန်အသံ" << endl;
    }
};

class Cat : public Animal {
public:
    void makeSound() override {  // Function overriding
        cout << "ပြောင်~" << endl;
    }
};

class Dog : public Animal {
public:
    void makeSound() override {  // Function overriding
        cout << "ဝါ~" << endl;
    }
};

int main() {
```

```
    Animal* animal1 = new Cat();
    Animal* animal2 = new Dog();

    animal1->makeSound();  // Cat ရဲ့ makeSound() ကို ခေါ်
    animal2->makeSound();  // Dog ရဲ့ makeSound() ကို ခေါ်

    delete animal1;
    delete animal2;
    return 0;
}
```

✓ Polymorphism ရဲ့ အကျိုးကျေးဇူးများ

1. Code Flexibility - ကုဒ်တွေကို ပိုမိုပြောင်းလွယ်ပြင်လွယ်ရှိစေပါတယ်
2. Code Reusability - ရှိပြီးသား code တွေကို ပြန်သုံးနိုင်ပါတယ်
3. Maintainability - ကုဒ်တွေကို ထိန်းသိမ်းရတာ ပိုမိုလွယ်ကူစေပါတယ်
4. Extensibility - လိုအပ်တဲ့အခါမှာ အသစ်တိုးချဲ့ရတာ လွယ်ကူစေပါတယ်

✓ Virtual Function နဲ့ Abstract Class

```
// Abstract class (interface သက်သက်)
class Shape {
public:
    virtual float area() = 0;  // Pure virtual function
    virtual float perimeter() = 0;
};

class Circle : public Shape {
private:
    float radius;
public:
    Circle(float r) : radius(r) {}
```

```
    float area() override {
        return 3.1416 * radius * radius;
    }

    float perimeter() override {
        return 2 * 3.1416 * radius;
    }
};

class Rectangle : public Shape {
private:
    float length, width;
public:
    Rectangle(float l, float w) : length(l), width(w) {}

    float area() override {
        return length * width;
    }

    float perimeter() override {
        return 2 * (length + width);
    }
};

int main() {
    Shape* shapes[2];
    shapes[0] = new Circle(5);
    shapes[1] = new Rectangle(4, 6);

    for(int i = 0; i < 2; i++) {
        cout << "Area: " << shapes[i]->area() << endl;
        cout << "Perimeter: " << shapes[i]->perimeter() << endl;
    }

    delete shapes[0];
    delete shapes[1];
    return 0;
}
```

❒ ABSTRACTION

Abstraction (အကျဉ်းချုံးခြင်း) ဆိုတာက OOP ရဲ့အရေးကြီးတဲ့ အခြေခံမူတစ်ခုဖြစ်ပြီး "အသေးစိတ် implementation တွေကို ဖုံးကွယ်ပြီး essential features တွေဘဲ ပြသခြင်း" လို့ အဓိပ္ပာယ်ဖွင့်ဆိုနိုင်ပါတယ်။

✓ Abstraction ရဲ့အဓိက ရည်ရွယ်ချက်များ

1. Complexity လျှော့ချခြင်း - အသုံးပြုသူအတွက် မလိုအပ်တဲ့ details တွေကို ဖုံးကွယ်ပေးပါတယ်
2. Code Maintainability - Implementation ပြောင်းလဲမှုတွေကို အလွယ်တကူလုပ်နိုင်ပါတယ်
3. Security - Internal data တွေကို ကာကွယ်ပေးပါတယ်
4. Reusability - Well-defined interfaces တွေကို ပြန်သုံးနိုင်ပါတယ်

✓ Abstraction ကို C++ မှာ အကောင်အထည်ဖော်နည်း

1. Abstract Class (အကျဉ်းချုံး Class) အသုံးပြုခြင်း

```
// Abstract class (Interface)
class Smartphone {
public:
    // Pure virtual functions (အကျဉ်းချုံး methods)
    virtual void makeCall(string number) = 0;
    virtual void sendSMS(string number, string message) = 0;
    virtual void takePhoto() = 0;

    // Normal method
    void powerOn() {
        cout << "ဖုန်းဖွင့်ပါတယ်..." << endl;
```

```
    }
};

// Concrete class (အကျဉ်းချုံး class ကို implement လုပ်ခြင်း)
class IPhone : public Smartphone {
public:
    void makeCall(string number) override {
        cout << "iPhone ဖြင့် " << number << " ကို ဖုန်းခေါ်နေပါတယ်" << endl;
    }

    void sendSMS(string number, string message) override {
        cout << "iPhone ဖြင့် " << number << " သို့ SMS ပို့နေပါတယ်: " << message << end
l;
    }

    void takePhoto() override {
        cout << "iPhone ဖြင့် ဓာတ်ပုံရိုက်နေပါတယ်" << endl;
    }
};

int main() {
    // Smartphone phone; // Error: Abstract class ကို instantiate မလုပ်နိုင်
    IPhone myPhone;

    myPhone.powerOn();
    myPhone.makeCall("09123456789");
    myPhone.takePhoto();

    return 0;
}
```

2. Interface အနေနဲ့ အသုံးပြုခြင်း

```
// Interface class (အကျဉ်းချုံး class)
class Database {
public:
    virtual void connect() = 0;
    virtual void query(string sql) = 0;
    virtual void disconnect() = 0;
```

```
};

// MySQL implementation
class MySQL : public Database {
public:
    void connect() override {
        cout << "MySQL နှင့် ချိတ်ဆက်နေပါတယ်..." << endl;
    }

    void query(string sql) override {
        cout << "MySQL တွင် query မေးနေပါတယ်: " << sql << endl;
    }

    void disconnect() override {
        cout << "MySQL နှင့် ချိတ်ဆက်မှု ဖြတ်နေပါတယ်" << endl;
    }
};

int main() {
    Database* db = new MySQL();

    db->connect();
    db->query("SELECT * FROM users");
    db->disconnect();

    delete db;
    return 0;
}
```

✓ Short Notes:

1. Implementation Details ကို ဖုံးကွယ်ပေးခြင်း

   User တွေက internal working တွေကို မသိဘဲ interface ဘဲ သုံးနိုင်ပါတယ်

2. Code ကို Module တွေအဖြစ် ခွဲခြားနိုင်ခြင်း

   အစိတ်အပိုင်းတစ်ခုချင်းစီကို သီးသန့် develop နိုင်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

3. Security ပိုမိုကောင်းမွန်ခြင်း

   Internal data တွေကို ပြင်ဆင်ခြင်းကနေ ကာကွယ်ပေးပါတယ်

4. Multiple Implementation လုပ်နိုင်ခြင်း

   တစ်ခုတည်းသော interface ကို အမျိုးမျိုးသော implementation တွေနဲ့ အသုံးပြုနိုင်ပါတယ်

Pure virtual function ဆိုတာ declaration သာပါပြီး definition မပါတဲ့ virtual function တစ်ခုဖြစ်ပါတယ်။ = 0 syntax ကိုသုံးပြီး သတ်မှတ်ပါတယ်။ ဒီ function ပါတဲ့ class ကို abstract class အဖြစ်သတ်မှတ်ပါတယ်။

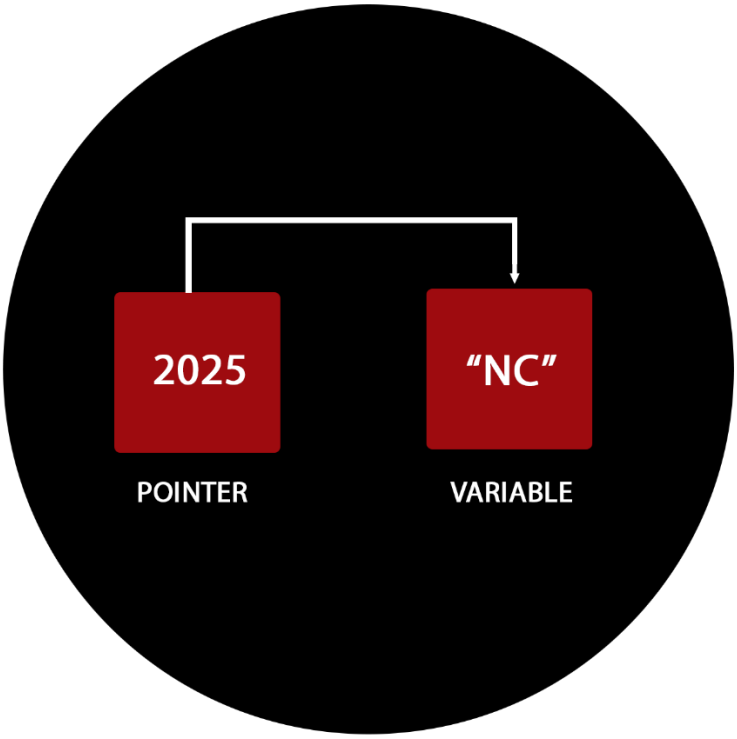
✓ လေ့ကျင့်ရန်

❒ Real-world Example - Banking System

```
// Abstract class for Bank Account
class BankAccount {
public:
    // Pure virtual functions
    virtual void deposit(double amount) = 0;
    virtual void withdraw(double amount) = 0;
    virtual double checkBalance() = 0;

    // Common functionality
    void displayAccountInfo(string accountNumber) {
        cout << "အကောင့်နံပါတ်: " << accountNumber << endl;
    }
};

// Concrete implementation
class SavingsAccount : public BankAccount {
```

```
private:
    double balance;
public:
    SavingsAccount() : balance(0) {}

    void deposit(double amount) override {
        if(amount > 0) {
            balance += amount;
            cout << amount << " ငွေထည့်ပြီးပါပြီ" << endl;
        }
    }

    void withdraw(double amount) override {
        if(amount > 0 && amount <= balance) {
            balance -= amount;
            cout << amount << " ငွေထုတ်ပြီးပါပြီ" << endl;
        } else {
            cout << "ငွေထုတ်ယူမှု မအောင်မြင်ပါ" << endl;
        }
    }

    double checkBalance() override {
        return balance;
    }
};

int main() {
    BankAccount* account = new SavingsAccount();
    account->displayAccountInfo("123456789");

    account->deposit(500000);
    account->withdraw(200000);
    cout << "လက်ကျန်ငွေ: " << account->checkBalance() << endl;


    delete account;
    return 0;
}
```

Pointers



❒ Pointers

Pointer ဆိုတာ variable တစ်ခုရဲ့ memory address ကို သိမ်းဆည်းထားတဲ့ special variable တစ်ခုဖြစ်ပါတယ်။ C++ မှာ pointer တွေကို * symbol သုံးပြီး declare လုပ်ပါတယ်။

✓ Pointer ၏ အခြေခံအသုံးပြုနည်း

1. Pointer Declaration နဲ့ Initialization

```
#include <iostream>
using namespace std;

int main() {
    int num = 10;       // Normal integer variable
    int* ptr = &num;    // Pointer variable that stores address of num

    cout << "Value of num: " << num << endl;
    cout << "Address of num: " << &num << endl;
    cout << "Value of ptr: " << ptr << endl;
    cout << "Value pointed by ptr: " << *ptr << endl;

    return 0;
}
```

2. Pointer Operations

```
int main() {
    int a = 5, b = 10;
    int* p1 = &a;
    int* p2 = &b;

    *p1 = 20;  // a ရဲ့တန်ဖိုးကို 20 ပြောင်းမယ်
    cout << "a = " << a << endl;  // 20

    p1 = p2;   // p1 ကို p2 ရဲ့ address ကိုညွှန်းခိုင်းမယ်
```

```
    cout << *p1 << endl;  // 10 (b ရဲ့တန်ဖိုး)

    return 0;
}
```

✓ Pointer အမျိုးအစားများ

1. Null Pointer

```
int* ptr = nullptr;  // Modern C++ style
// or int* ptr = NULL;  // Older style
// or int* ptr = 0;
```

2. Wild Pointer (အန္တရာယ်ရှိသော pointer)

```
int* ptr;  // Uninitialized pointer (wild pointer)
// cout << *ptr;  // Dangerous! Undefined behavior
```

3. Dangling Pointer (မှားယွင်းသော reference)

```
int* func() {
    int x = 10;
    return &x;  // x က function ပြီးတာနဲ့ပျက်သွားမယ်
}

int main() {
    int* p = func();
    // cout << *p;  // Dangling pointer - undefined behavior
    return 0;
}
```

✓ Pointer Arithmetic

```
int main() {
```

```
    int arr[] = {10, 20, 30, 40, 50};
    int* ptr = arr;  // arr က array ရဲ့ပထမဆုံး element ကိုညွှန်းတယ်

    cout << *ptr << endl;    // 10
    cout << *(ptr + 1) << endl;  // 20 (next element)
    cout << *(ptr + 3) << endl;  // 40

    ptr += 2;  // Pointer ကို 2 ခွေ့မယ်
    cout << *ptr << endl;    // 30

    return 0;
}
```

Pointer နဲ့ Array

```
int main() {
    double numbers[] = {1.1, 2.2, 3.3, 4.4, 5.5};
    double* p = numbers;

    for(int i = 0; i < 5; i++) {
        cout << "Value at index " << i << ": " << *(p + i) << endl;
        // or cout << p[i] << endl;  // Same as array access
    }

    return 0;
}
```

✓ Pointer နဲ့ Function

1. Pass by Pointer

```
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
```

```
    int x = 5, y = 10;
    swap(&x, &y);
    cout << "x = " << x << ", y = " << y << endl;  // x=10, y=5
    return 0;
}
```

2. Return Pointer from Function

```
int* createArray(int size) {
    int* arr = new int[size];  // Dynamic memory allocation
    for(int i = 0; i < size; i++) {
        arr[i] = i * 10;
    }
    return arr;
}

int main() {
    int* myArray = createArray(5);
    for(int i = 0; i < 5; i++) {
        cout << myArray[i] << " ";  // 0 10 20 30 40
    }
    delete[] myArray;  // Memory ကို free လုပ်မယ်
    return 0;
}
```

✓ Smart Pointers (Modern C++)

1. unique_ptr

```
#include <memory>

int main() {
    unique_ptr<int> ptr(new int(50));
    cout << *ptr << endl;  // 50
    // Memory automatically freed when ptr goes out of scope
    return 0;
}
```

2. shared_ptr

```
shared_ptr<int> ptr1(new int(100));
{
    shared_ptr<int> ptr2 = ptr1;  // Both point to same memory
    cout << *ptr1 << " " << *ptr2 << endl;  // 100 100
}  // ptr2 က scope ပြင်ပရောက်တာနဲ့ destroy ဖြစ်မယ်

cout << *ptr1 << endl;  // 100 (ptr1 ကတော့အလုပ်လုပ်ဆဲ)
```

✓ Pointer သုံးရတဲ့ အကြောင်းရင်းများ

1. Dynamic Memory Allocation - Runtime မှာ memory လိုအပ်သလောက်ယူဖို့
2. Arrays and Strings - Array တွေကို efficient ဖြစ်အောင် access လုပ်ဖို့
3. Function Arguments - Large data တွေကို copy မလုပ်ဘဲ pass လုပ်ဖို့
4. Data Structures - Linked lists, trees, graphs တွေတည်ဆောက်ဖို့
5. System Programming - Hardware resources တွေကို directly access လုပ်ဖို့

✓ Pointer သုံးရာမှာ သတိထားရမယ့်အချက်များ

1. Always Initialize Pointers - Wild pointer တွေမဖြစ်အောင်
2. Check for nullptr - Dereference မလုပ်ခင် null ဟုတ်မဟုတ်စစ်ပါ
3. Memory Leaks - new နဲ့ allocate လုပ်ထားရင် delete လုပ်ဖို့မမေ့ပါနဲ့
4. Dangling Pointers - Free လုပ်ပြီးသော memory ကို access မလုပ်မိပါစေနဲ့
5. Pointer Arithmetic - Array bounds တွေကိုမကျော်မိအောင်ဂရုစိုက်ပါ

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန်

ဒီ program မှာ user ကနေ number နှစ်ခုရိုက်ထည့်ပြီး၊ pass by reference နည်းလမ်းကိုသုံးပြီး ပေါင်းခြင်း၊ နုတ်ခြင်း operation တွေကို လုပ်ဆောင်ပြမယ်။

```
#include <iostream>
using namespace std;

// Function prototypes
void addNumbers(int &a, int &b, int &result);
void subtractNumbers(int &a, int &b, int &result);
void displayResult(const char* operationName, int &result);

int main() {
    int input1, input2;
    int sumResult, diffResult;

    cout << "Enter Num1:";
    cin >> input1;

    cout << "Enter Num2:";
    cin >> input2;


    // Addition operation using pass by reference
    addNumbers(input1, input2, sumResult);
    displayResult("The sum is :", sumResult);


    // Subtraction operation using pass by reference
    subtractNumbers(input1, input2, diffResult);
    displayResult("The difference is :", diffResult);


    return 0;
}

// Function to add two numbers using references
void addNumbers(int &a, int &b, int &result) {
    result = a + b;
```

```
}

// Function to subtract two numbers using references
void subtractNumbers(int &a, int &b, int &result) {
    result = a - b;
}

// Function to display the result
void displayResult(const char* operationName, int &result) {
    cout << operationName << ": " << result << endl;
}
```

✓ Program အလုပ်လုပ်ပုံ

1. Input ရယူခြင်း:
    - User ကနေ integer နှစ်ခုကို space ခြားပြီး ရိုက်ထည့်မယ်
2. Pass by Reference အသုံးပြုခြင်း:
    - addNumbers နဲ့ subtractNumbers function တွေမှာ parameter တွေကို reference အနေနဲ့လက်ခံထားတယ်
    - ရလာဒ်ကို result parameter ကနေတဆင့် return လုပ်မယ့်အစား reference အနေနဲ့ ပြောင်းလဲပေးမယ်
3. Addition Operation:
    - addNumbers(input1, input2, sumResult); လို့ call လုပ်တဲ့အခါ
    - input တွေကို reference အနေနဲ့ပို့ပြီး ရလာဒ်ကို sumResult မှာသိမ်းမယ်
4. Subtraction Operation:
    - subtractNumbers(input1, input2, diffResult); လို့ call လုပ်တဲ့အခါ
    - input တွေကို reference အနေနဲ့ပို့ပြီး ရလာဒ်ကို diffResult မှာသိမ်းမယ်

5. Output ပြခြင်း:
    - displayResult function က ရလာဒ်တွေကို operation name နဲ့တကွ console မှာ ပြပေးတယ်

Output:

```
Enter Num1: 25
Enter Num2:10

The sum is : 35
The difference is :15
```

✓ ရှင်းလင်းချက်

1. Memory Efficient:
    - Reference ကိုသုံးတဲ့အတွက် variable အသစ်တွေမဖန်တီးဘဲ original variable တွေကိုတိုက်ရိုက်အသုံးပြုတယ်
2. Modify Original Values:
    - Function ထဲမှာ parameter တွေရဲ့တန်ဖိုးကိုပြောင်းလဲနိုင်တယ်
3. Multiple Return Values:
    - Return value တစ်ခုထက်ပိုပြီး reference parameter တွေကနေတဆင့် ရလာဒ်တွေကို return လုပ်နိုင်တယ်
4. Avoid Copying:
    - Large data structures တွေကို pass လုပ်တဲ့အခါ copy မလုပ်ဘဲ reference အနေနဲ့သုံးတဲ့အတွက် performance ပိုကောင်းတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Header Files

❒ Header Files

C++ မှာ header files (ခေါင်းစီးဖိုင်များ) ဆိုတာက source code တွေကို စနစ်တကျ စီမံခန့်ခွဲဖို့ အသုံးပြုတဲ့ .h (သို့) .hpp extension ပါတဲ့ဖိုင်တွေဖြစ်ပါတယ်။ Header files တွေမှာ class declarations, function prototypes, constants, macros တွေကို သိမ်းဆည်းထားလေ့ရှိပါတယ်။

✓ Header Files အသုံးပြုရတဲ့ အကြောင်းရင်းများ

1. Code Organization - ကုဒ်တွေကို ပိုမိုစနစ်ကျစွာ စီမံနိုင်ပါတယ်
2. Reusability - တစ်ခါရေးပြီး အကြိမ်ကြိမ်ပြန်သုံးနိုင်ပါတယ်
3. Compilation Time - ပြောင်းလဲမှုတွေရှိရင် အချိန်ကုန်သက်သာစေပါတယ်
4. Namespace Pollution - Global namespace ကို မဖျက်ဆီးပါဘူး

✓ Header File တစ်ခု၏ အခြေခံ structure

```
#ifndef HEADER_NAME_H  // Header guard (အောက်တွင်ရှင်းပြထားပါတယ်)
#define HEADER_NAME_H

// System headers
#include <iostream>
#include <string>

// Other custom headers
#include "MyClass.h"

// Constants
const double PI = 3.14159;

// Macros
#define MAX(a,b) ((a) > (b) ? (a) : (b))

// Function declarations
```

```
void printMessage(const std::string& msg);

// Class declarations
class MyClass {
    // Class members...
};

#endif // HEADER_NAME_H
```

✓ Header Guards ဆိုတာဘာလဲ?

Header guard (သို့) include guard ဆိုတာက header file တစ်ခုကို တစ်ကြိမ်ထက်ပိုပြီး include လုပ်မိတဲ့အခါ multiple definition errors တွေမဖြစ်အောင် ကာကွယ်ပေးပါတယ်။

ဥပမာ:

```
#ifndef MY_HEADER_H
#define MY_HEADER_H

// Header content...

#endif // MY_HEADER_H
```

✓ Standard Library Headers vs Custom Headers

1. Standard Library Headers

```
#include <iostream>    // Angle brackets သုံးပါတယ်
#include <vector>
#include <string>
```

2. Custom Headers

```
#include "myheader.h"  // Double quotes သုံးပါတယ်
```

```
#include "../includes/utility.h"
```

✓ ကောင်းမွန်တဲ့ Header File ရေးသားနည်း

1. Self-Contained - အခြား header တွေမလိုဘဲ အလုပ်လုပ်နိုင်ရမယ်
2. Minimal Dependencies - လိုအပ်တဲ့ header တွေဘဲ include လုပ်ပါ
3. Header Guards - အမြဲတမ်း header guards သုံးပါ
4. Forward Declarations - လိုအပ်ရင် forward declarations သုံးပါ
5. Namespace Usage - Namespace တွေကို သေချာစီမံပါ

✓ Forward Declarations အကြောင်း

Header file တွေမှာ class တွေရဲ့ full definition မလိုဘဲ declaration ဘဲလိုတဲ့အခါ forward declaration သုံးပါတယ်။

```
// Forward declaration
class MyClass;

void processMyClass(MyClass* obj);  // MyClass ရဲ့ full definition မလိုပါ
```

✓ Inline Functions in Headers

Header files တွေမှာ inline functions တွေကို သတ်မှတ်လေ့ရှိပါတယ်။

```
inline int square(int x) {
    return x * x;
}
```

✓ Templates in Headers

Template definitions တွေကို header files တွေထဲမှာဘဲ ရေးရပါတယ်။

```
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}
```

✓ Common C++ Standard Library Headers

1. I/O Operations
    - <iostream> - Basic input/output operations
    - <fstream> - File operations

2. Containers
    - <vector> - Dynamic array
    - <map> - Associative container
    - <string> - String class
    - Algorithms
    - <algorithm> - Common algorithms

3. Utilities
    - <memory> - Smart pointers
    - <chrono> - Time utilities

✓ Short Notes:

1. Include What You Use - လိုအပ်တဲ့ header တွေဘဲ include လုပ်ပါ
2. Avoid Nested Includes - Header တွေအများကြီး nested မလုပ်ပါနဲ့
3. Logical Grouping - ဆက်စပ်တဲ့ declarations တွေကို တူတဲ့ header ထဲမှာဘဲ ထားပါ
4. Documentation - Header files တွေကို ကောင်းကောင်း documentation ရေးပါ

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန် -

❒ Student Management System

```
StudentManagementSystem/
├─── main.cpp                (Main application file)
├─── Student.h               (Student class header)
├─── Student.cpp             (Student class implementation)
├─── StudentManager.h        (Student management header)
├─── StudentManager.cpp      (Student management implementation)
└─── student_records.txt     (Student records text file)
```

Student Class

⇨ Student.h

```
#ifndef STUDENT_H
#define STUDENT_H

#include <string>
#include <iostream>

class Student {
private:
    int id;
    std::string name;
    std::string course;
    std::string phone;

public:
    Student(int id = 0, std::string name = "", std::string course = "", std::
string phone = "");

    // Getters
    int getId() const;
    std::string getName() const;
```

```
    std::string getCourse() const;
    std::string getPhone() const;

    // Setters
    void setId(int id);
    void setName(std::string name);
    void setCourse(std::string course);
    void setPhone(std::string phone);

    void display() const;
    std::string toFileString() const;
    static Student fromFileString(const std::string& data);
};

#endif
```

⇨ Student.cpp

```
#include "Student.h"
#include <sstream>

Student::Student(int id, std::string name, std::string course, std::string ph
one)
    : id(id), name(name), course(course), phone(phone) {}

// Getters
int Student::getId() const { return id; }
std::string Student::getName() const { return name; }
std::string Student::getCourse() const { return course; }
std::string Student::getPhone() const { return phone; }

// Setters
void Student::setId(int id) { this->id = id; }
void Student::setName(std::string name) { this->name = name; }
void Student::setCourse(std::string course) { this->course = course; }
void Student::setPhone(std::string phone) { this->phone = phone; }

void Student::display() const {
    std::cout << "ID: " << id << "\nName: " << name
              << "\nCourse: " << course << "\nPhone: " << phone << "\n\n";
}
```

```
std::string Student::toFileString() const {
    return std::to_string(id) + "|" + name + "|" + course + "|" + phone;
}

Student Student::fromFileString(const std::string& data) {
    std::istringstream iss(data);
    std::string token;
    std::vector<std::string> tokens;

    while (getline(iss, token, '|')) {
        tokens.push_back(token);
    }

    if (tokens.size() == 4) {
        return Student(std::stoi(tokens[0]), tokens[1], tokens[2], tokens[3])
;
    }
    return Student();
}
```

⇨ Student Manager Class

StudentManager.h

```
#ifndef STUDENTMANAGER_H
#define STUDENTMANAGER_H

#include "Student.h"
#include <vector>
#include <fstream>

class StudentManager {
private:
    const std::string filename = "student_records.txt";

public:
```

```
    void addStudent(const Student& student);
    std::vector<Student> getAllStudents();
    bool updateStudentCourse(int id, const std::string& newCourse);
    bool deleteStudent(int id);
};

#endif
```

⇨ StudentManager.cpp

```
#include "StudentManager.h"
#include <iostream>
#include <algorithm>

void StudentManager::addStudent(const Student& student) {
    std::ofstream outFile(filename, std::ios::app);
    if (outFile) {
        outFile << student.toFileString() << "\n";
    } else {
        std::cerr << "Error opening file for writing!\n";
    }
    outFile.close();
}

std::vector<Student> StudentManager::getAllStudents() {
    std::vector<Student> students;
    std::ifstream inFile(filename);

    if (inFile) {
        std::string line;
        while (getline(inFile, line)) {
            students.push_back(Student::fromFileString(line));
        }
    } else {
        std::cerr << "Error opening file for reading!\n";
    }
    inFile.close();
    return students;
}
```

```
bool StudentManager::updateStudentCourse(int id, const std::string& newCourse
) {
    std::vector<Student> students = getAllStudents();
    bool found = false;

    for (auto& student : students) {
        if (student.getId() == id) {
            student.setCourse(newCourse);
            found = true;
            break;
        }
    }

    if (found) {
        std::ofstream outFile(filename);
        if (outFile) {
            for (const auto& student : students) {
                outFile << student.toFileString() << "\n";
            }
        } else {
            std::cerr << "Error opening file for updating!\n";
            return false;
        }
        outFile.close();
    }

    return found;
}

bool StudentManager::deleteStudent(int id) {
    std::vector<Student> students = getAllStudents();
    auto it = std::remove_if(students.begin(), students.end(),
        [id](const Student& s) { return s.getId() == id; });

    if (it != students.end()) {
        students.erase(it, students.end());

        std::ofstream outFile(filename);
        if (outFile) {
            for (const auto& student : students) {
                outFile << student.toFileString() << "\n";
            }
```

```
        } else {
            std::cerr << "Error opening file for deletion!\n";
            return false;
        }
        outFile.close();
        return true;
    }

    return false;
}
```

✓ Main Application

main.cpp

```
#include <iostream>
#include "Student.h"
#include "StudentManager.h"

void displayMenu() {
    std::cout << "\nStudent Management System\n";
    std::cout << "1. Add New Student\n";
    std::cout << "2. View All Students\n";
    std::cout << "3. Update Student Course\n";
    std::cout << "4. Delete Student\n";
    std::cout << "5. Exit\n";
    std::cout << "Enter your choice: ";
}

int getIntInput(const std::string& prompt) {
    int value;
    std::cout << prompt;
    while (!(std::cin >> value)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "Invalid input. Please enter a number: ";
    }
    std::cin.ignore();
    return value;
}

std::string getStringInput(const std::string& prompt) {
```

```
    std::string value;
    std::cout << prompt;
    std::getline(std::cin, value);
    return value;
}

int main() {
    StudentManager manager;
    int choice;

    do {
        displayMenu();
        choice = getIntInput("");

        switch (choice) {
            case 1: {
                int id = getIntInput("Enter Student ID: ");
                std::string name = getStringInput("Enter Student Name: ");
                std::string course = getStringInput("Enter Course: ");
                std::string phone = getStringInput("Enter Phone: ");

                Student newStudent(id, name, course, phone);
                manager.addStudent(newStudent);
                std::cout << "Student added successfully!\n";
                break;
            }
            case 2: {
                std::vector<Student> students = manager.getAllStudents();
                if (students.empty()) {
                    std::cout << "No student records found.\n";
                } else {
                    std::cout << "\n--- Student Records ---\n";
                    for (const auto& student : students) {
                        student.display();
                    }
                }
                break;
            }
            case 3: {
                int id = getIntInput("Enter Student ID to update: ");
                std::string newCourse = getStringInput("Enter New Course: ");
```

```
                if (manager.updateStudentCourse(id, newCourse)) {
                    std::cout << "Student course updated successfully!\n";
                } else {
                    std::cout << "Student with ID " << id << " not found.\n";
                }
                break;
            }
            case 4: {
                int id = getIntInput("Enter Student ID to delete: ");

                if (manager.deleteStudent(id)) {
                    std::cout << "Student deleted successfully!\n";
                } else {
                    std::cout << "Student with ID " << id << " not found.\n";
                }
                break;
            }
            case 5:
                std::cout << "Exiting program...\n";
                break;
            default:
                std::cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 5);

    return 0;
}
```

✓ Testing Results

Test Case 1: Adding New Students

```
Student Management System
1. Add New Student
2. View All Students
3. Update Student Course
4. Delete Student
5. Exit
Enter your choice: 1
Enter Student ID: 101
```

```
Enter Student Name: မောင်မောင်
Enter Course: C++ Programming
Enter Phone: 09123456789
Student added successfully!
```

Test Case 2: Viewing All Students

```
--- Student Records ---
ID: 101
Name: မောင်မောင်
Course: C++ Programming
Phone: 09123456789
```

Test Case 3: Updating Student Course

```
Enter Student ID to update: 101
Enter New Course: Advanced C++
Student course updated successfully!
```

Test Case 4: Deleting Student

```
Enter Student ID to delete: 101
Student deleted successfully!
```

STL Containers

❒ STL Container

C++ Standard Template Library (STL) မှာ container တွေဆိုတာ data တွေကို စုစည်းဖို့ အသုံးပြုတဲ့ template classes တွေဖြစ်ပါတယ်။ Container တွေက memory management နဲ့ data organization တွေကို အလိုအလျောက် စီမံပေးပါတယ်။

✓ Container အမျိုးအစားများ

1. Sequence Containers (အစဉ်လိုက်ထားသော container များ)

vector (dynamic array)

```
#include <vector>
#include <iostream>

int main() {
    std::vector<int> numbers = {1, 2, 3};

    // နောက်ဆုံးမှာ element ထည့်ခြင်း
    numbers.push_back(4);  // {1, 2, 3, 4}

    // Index နဲ့ access လုပ်ခြင်း
    std::cout << "Element at index 2: " << numbers[2] << '\n';

    // Range-based for loop
    for (int num : numbers) {
        std::cout << num << ' ';
    }

    return 0;
}
```

list (Doubly-linked list)

```
#include <list>
#include <iostream>
```

```
int main() {
    std::list<std::string> names = {"Alice", "Bob"};

    // ရှေ့မှာ element ထည့်ခြင်း
    names.push_front("Eve");  // {"Eve", "Alice", "Bob"}

    // Iterator သုံးပြီး traverse လုပ်ခြင်း
    for (auto it = names.begin(); it != names.end(); ++it) {
        std::cout << *it << '\n';
    }

    return 0;
}
```

deque (Double-ended queue)

```
#include <deque>
#include <iostream>

int main() {
    std::deque<int> dq = {10, 20, 30};

    // ရှေ့နောက် element တွေ ထည့်ခြင်း
    dq.push_front(5);   // {5, 10, 20, 30}
    dq.push_back(40);   // {5, 10, 20, 30, 40}

    std::cout << "Front: " << dq.front() << ", Back: " << dq.back() << '\n';

    return 0;
}
```

2. Associative Containers (အညွှန်းနဲ့ထားသော container များ)

set (Unique elements, sorted order)

```
#include <set>
#include <iostream>

int main() {
    std::set<int> uniqueNumbers = {5, 2, 8, 2, 5};  // {2, 5, 8}

    // Insert
    uniqueNumbers.insert(3);  // {2, 3, 5, 8}

    // Find
    if (uniqueNumbers.find(5) != uniqueNumbers.end()) {
        std::cout << "5 တွေ့ပြီ\n";
    }

    return 0;
}
```

map (Key-value pairs, sorted by key)

```
#include <map>
#include <iostream>

int main() {
    std::map<std::string, int> ages = {
        {"Alice", 25},
        {"Bob", 30}
    };

    // Insert
    ages["Charlie"] = 28;

    // Access
    std::cout << "Alice's age: " << ages["Alice"] << '\n';

    // Iterate
    for (const auto& pair : ages) {
```

```
        std::cout << pair.first << " is " << pair.second << " years old\n";
    }

    return 0;
}
```

3. Unordered Containers (Hash-based containers)

unordered_set (Unique elements, hashed)

```
#include <unordered_set>
#include <iostream>

int main() {
    std::unordered_set<std::string> names = {"Alice", "Bob"};

    // Insert
    names.insert("Charlie");

    // Check existence
    if (names.count("Alice") > 0) {
        std::cout << "Alice ရှိပါတယ်\n";
    }

    return 0;
}
```

unordered_map (Key-value pairs, hashed)

```
#include <unordered_map>
#include <iostream>

int main() {
    std::unordered_map<std::string, std::string> capitals = {
        {"Myanmar", "Naypyitaw"},
        {"Thailand", "Bangkok"}
    };

    // Insert
```

```
    capitals["Japan"] = "Tokyo";

    // Access
    std::cout << "Myanmar's capital: " << capitals["Myanmar"] << '\n';

    return 0;
}
```

✓ Container Adaptors (Container အထောက်အကူများ)

stack (LIFO - Last In First Out)

```
#include <stack>
#include <iostream>

int main() {
    std::stack<int> s;

    s.push(10);  // Bottom
    s.push(20);
    s.push(30);  // Top

    while (!s.empty()) {
        std::cout << s.top() << ' ';  // 30 20 10
        s.pop();
    }

    return 0;
}
```

queue (FIFO - First In First Out)

```
#include <queue>
#include <iostream>

int main() {
    std::queue<std::string> q;

    q.push("First");
    q.push("Second");
    q.push("Third");

    while (!q.empty()) {
        std::cout << q.front() << ' ';  // First Second Third
        q.pop();
    }

    return 0;
}
```

✓ Container Selection Guide

1. Random access လိုရင် → vector, deque
2. Frequent insert/delete at both ends → deque, list
3. Middle မှာ frequent insert/delete → list
4. Key-value pairs with sorted order → map
5. Key-value pairs with fast lookup → unordered_map
6. Unique elements with sorted order → set
7. Unique elements with fast lookup → unordered_set

✓ Common Container Operations

1. Iteration (အားလုံးသော container များအတွက်)

```
std::vector<int> vec = {1, 2, 3};

// Range-based for loop (C++11)
for (int num : vec) {
    std::cout << num << ' ';
}

// Iterator သုံးပြီး
for (auto it = vec.begin(); it != vec.end(); ++it) {
    std::cout << *it << ' ';
}
```

2. Size Operations

```
std::list<double> values = {1.1, 2.2, 3.3};

if (!values.empty()) {
    std::cout << "Size: " << values.size() << '\n';
}
```

3. Element Access

```
std::array<int, 3> arr = {10, 20, 30};

// Random access
std::cout << arr[1] << '\n';  // 20

// Safe access with bounds checking
std::cout << arr.at(1) << '\n';  // 20
```

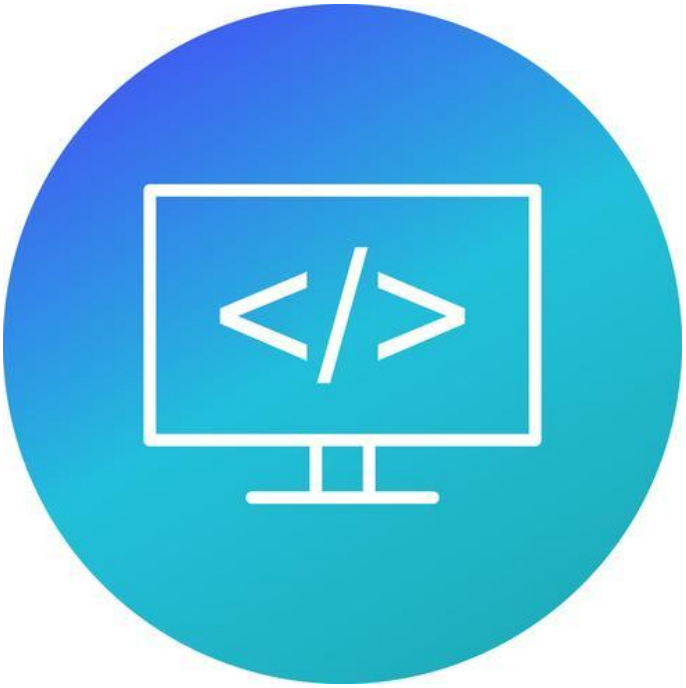
✓ Performance Comparison

Operation	vector	deque	list	set	map	unordered_set
Insert at end	O(1)	O(1)	O(1)	-	-	-
Insert at middle	O(n)	O(n)	O(1)	O(log n)	O(log n)	O(1) avg
Random access	O(1)	O(1)	O(n)	-	-	-
Search	O(n)	O(n)	O(n)	O(log n)	O(log n)	O(1) avg
Memory usage	Low	Medium	High	High	High	Medium

C++ containers တွေကို ကောင်းစွာအသုံးပြုခြင်းဖြင့် သင့် program တွေကို ပိုမိုထိရောက်စွာ ရေးသားနိုင်မှာဖြစ်ပါတယ်။ Container တစ်ခုချင်းစီရဲ့အားသာချက်၊ အားနည်းချက်တွေကို နားလည်ပြီး သင့်တော်ရာ container ကို ရွေးချယ်အသုံးပြုသင့်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Vector



❒ Vector

C++ မှာ vector ဆိုတာ Standard Template Library (STL) ရဲ့ dynamic array container တစ်ခုဖြစ်ပါတယ်။ Vector က size ကို အလိုအလျောက် resize လုပ်ပေးနိုင်ပြီး element တွေကို contiguous memory location မှာ သိမ်းဆည်းပေးပါတယ်။

✓ Vector ၏ အခြေခံအသုံးပြုနည်း

1. Vector ကို Initialize လုပ်ခြင်း

```
#include <vector>
#include <iostream>

int main() {
    // ဗလာ vector တစ်ခု ဖန်တီးခြင်း
    std::vector<int> numbers1;

    // Size နဲ့ initialize လုပ်ခြင်း (default value 0)
    std::vector<int> numbers2(5);  // {0, 0, 0, 0, 0}

    // Size နဲ့ initial value သတ်မှတ်ခြင်း
    std::vector<int> numbers3(3, 10);  // {10, 10, 10}

    // Initializer list သုံးခြင်း
    std::vector<int> numbers4 = {1, 2, 3, 4, 5};

    // Array တစ်ခုကနေ copy လုပ်ခြင်း
    int arr[] = {6, 7, 8};
    std::vector<int> numbers5(arr, arr + 3);  // {6, 7, 8}

    return 0;
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

2. Vector ထဲကို Element တွေ ထည့်ခြင်း

```
std::vector<std::string> names;

// နောက်ဆုံးမှာ element ထည့်ခြင်း
names.push_back("Alice");
names.push_back("Bob");

// ရှေ့မှာ element ထည့်ခြင်း (inefficient for vector)
names.insert(names.begin(), "Eve");

// Specific position မှာ ထည့်ခြင်း
names.insert(names.begin() + 1, "Charlie");
```

3. Vector ထဲက Element တွေကို Access လုပ်ခြင်း

```
std::vector<double> prices = {10.5, 20.3, 30.2};

// Index သုံးပြီး access လုပ်ခြင်း (no bounds checking)
double first = prices[0];

// at() function သုံးခြင်း (bounds checking ပါတယ်)
double second = prices.at(1);

// Front နဲ့ back elements
double front = prices.front();  // 10.5
double back = prices.back();    // 30.2

// Iterator သုံးပြီး access လုပ်ခြင်း
for (auto it = prices.begin(); it != prices.end(); ++it) {
    std::cout << *it << ' ';
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Vector ၏ အရေးကြီးသော Member Functions

1. Size နဲ့ Capacity ဆိုင်ရာ Functions

```
std::vector<int> vec = {1, 2, 3};

// Vector ရဲ့current size
std::cout << "Size: " << vec.size() << '\n';  // 3

// Vector ရဲ့maximum possible size
std::cout << "Max size: " << vec.max_size() << '\n';

// Vector က ဗလာလား စစ်ခြင်း
if (vec.empty()) {
    std::cout << "Vector is empty\n";
}

// Vector ရဲ့current capacity
std::cout << "Capacity: " << vec.capacity() << '\n';

// Capacity ကို ပြောင်းခြင်း
vec.reserve(100);  // Capacity ကို100 လုပ်မယ်
```

2. Element တွေကို ဖျက်ခြင်း

```
std::vector<char> letters = {'a', 'b', 'c', 'd', 'e'};

// နောက်ဆုံးelement ကို ဖျက်ခြင်း
letters.pop_back();  // {'a', 'b', 'c', 'd'}

// Specific position ကelement ကို ဖျက်ခြင်း
letters.erase(letters.begin() + 1);  // {'a', 'c', 'd'}

// Range of elements တွေကို ဖျက်ခြင်း
```

```
letters.erase(letters.begin(), letters.begin() + 2);  // {'d'}

// Vector ထဲက element အားလုံးကို ဖျက်ခြင်း
letters.clear();  // {}
```

Vector ၏ အထူးခြားသော Features

1. Dynamic Resizing

```
std::vector<int> nums;
nums.reserve(3);  // Capacity ကို 3 လုပ်မယ်


nums.push_back(1);
nums.push_back(2);
nums.push_back(3);

// Capacity ပြည့်သွားရင် အလိုအလျောက် resize လုပ်မယ်
nums.push_back(4);  // Capacity ကို 2 ဆ (6) လုပ်မယ်


std::cout << "Size: " << nums.size() << '\n';      // 4
std::cout << "Capacity: " << nums.capacity() << '\n';  // 6
```

2. Range-based for loop (C++11)

```
std::vector<std::string> fruits = {"Apple", "Banana", "Orange"};

for (const auto& fruit : fruits) {
    std::cout << fruit << '\n';
}
```

3. Vector ကို Copy နဲ့ Move လုပ်ခြင်း

```
std::vector<int> original = {1, 2, 3};

// Copy constructor
```

```
std::vector<int> copy = original;

// Move constructor (C++11)
std::vector<int> moved = std::move(original);  // original က ဗလာဖြစ်သွားမယ်
```

Vector ကို အသုံးပြုရမယ့် အခြေအနေများ

1. Random access လိုအပ်တဲ့အခါ - O(1) complexity
2. Element တွေကို အဆုံးမှာ ထည့်/ဖျက်တဲ့အခါ - O(1) amortized
3. Memory ကို contiguous လိုအပ်တဲ့အခါ (e.g., array လိုအခြေအနေမျိုး)
4. Size ကို ကြိုမသိတဲ့အခါ

Vector မသုံးသင့်တဲ့ အခြေအနေများ

1. Frequent insert/delete at beginning/middle - O(n) complexity
2. Fixed size ရှိတဲ့အခါ - std::array က ပိုသင့်တော်ပါတယ်
3. Very large sizes အတွက် - Memory fragmentation ဖြစ်နိုင်ပါတယ်

Performance Tips

1. reserve() ကို အသုံးပြုပါ - Push_back များများလုပ်မယ်ဆိုရင် ကြိုတင်သတ်မှတ်ထားပါ
2. shrink_to_fit() - Extra capacity တွေကို လျှော့ချဖို့
3. emplace_back() - Push_back ထက် ပိုထိရောက်ပါတယ် (C++11)
4. data() - Underlying array ကို direct access လုပ်ဖို့

```
std::vector<int> vec;
vec.reserve(100);  // ကြိုတင်နေရာယူထားခြင်း

for (int i = 0; i < 100; ++i) {
    vec.emplace_back(i);  // Efficient construction
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
vec.shrink_to_fit();  // Extra capacity တွေကို ဖယ်ရှားမယ်

// Raw array pointer ရခြင်း
int* arr = vec.data();
```

Vector က C++ မှာ အသုံးအများဆုံး container တစ်ခုဖြစ်ပြီး ကောင်းစွာအသုံးပြုနိုင်ရင် သင့် program တွေကို ပိုမိုထိရောက်စွာ ရေးသားနိုင်မှာဖြစ်ပါတယ်။

လေ့ကျင့်ရန် -

❒ Students CRUD Application using Vector

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

class Student {
private:
    int id;
    std::string name;
    std::string major;
    int age;

public:
    Student(int id, std::string name, std::string major, int age)
        : id(id), name(name), major(major), age(age) {}

    // Getters
    int getId() const { return id; }
```

```
    std::string getName() const { return name; }
    std::string getMajor() const { return major; }
    int getAge() const { return age; }

    // Setters
    void setName(std::string newName) { name = newName; }
    void setMajor(std::string newMajor) { major = newMajor; }
    void setAge(int newAge) { age = newAge; }

    // Display student information
    void display() const {
        std::cout << "\nID: " << id
                  << "\nName: " << name
                  << "\nMajor: " << major
                  << "\nAge: " << age << "\n";
    }
};

class StudentManager {
private:
    std::vector<Student> students;
    int nextId = 1;

    // Helper function to find student by ID
    std::vector<Student>::iterator findStudent(int id) {
        return std::find_if(students.begin(), students.end(),
            [id](const Student& s) { return s.getId() == id; });
    }

public:
    // Create a new student
    void addStudent(std::string name, std::string major, int age) {
        students.emplace_back(nextId++, name, major, age);
        std::cout << "Student added successfully!\n";
    }

    // Read all students
    void displayAllStudents() const {
        if (students.empty()) {
            std::cout << "No students found.\n";
            return;
        }
```

```
        std::cout << "\n=== Student List ===\n";
        for (const auto& student : students) {
            student.display();
        }
    }

    // Read single student
    void displayStudent(int id) const {
        auto it = std::find_if(students.begin(), students.end(),
            [id](const Student& s) { return s.getId() == id; });

        if (it != students.end()) {
            it->display();
        } else {
            std::cout << "Student with ID " << id << " not found.\n";
        }
    }

    // Update student information
    void updateStudent(int id, std::string newName, std::string newMajor, int
newAge) {
        auto it = findStudent(id);
        if (it != students.end()) {
            it->setName(newName);
            it->setMajor(newMajor);
            it->setAge(newAge);
            std::cout << "Student updated successfully!\n";
        } else {
            std::cout << "Student with ID " << id << " not found.\n";
        }
    }

    // Delete a student
    void deleteStudent(int id) {
        auto it = findStudent(id);
        if (it != students.end()) {
            students.erase(it);
            std::cout << "Student deleted successfully!\n";
        } else {
            std::cout << "Student with ID " << id << " not found.\n";
        }
```

```
    }
};

// User interface functions
void displayMenu() {
    std::cout << "\n=== Student Management System ===\n"
              << "1. Add New Student\n"
              << "2. View All Students\n"
              << "3. View Student by ID\n"
              << "4. Update Student\n"
              << "5. Delete Student\n"
              << "6. Exit\n"
              << "Enter your choice: ";
}

std::string getStringInput(const std::string& prompt) {
    std::string input;
    std::cout << prompt;
    std::getline(std::cin >> std::ws, input);  // ws to skip any leftover newl
ines
    return input;
}

int getIntInput(const std::string& prompt) {
    int value;
    std::cout << prompt;
    while (!(std::cin >> value)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "Invalid input. Please enter a number: ";
    }
    std::cin.ignore();  // Ignore leftover newline
    return value;
}

int main() {
    StudentManager manager;
    int choice;

    do {
        displayMenu();
        choice = getIntInput("");
```

```
        switch (choice) {
            case 1: {
                std::string name = getStringInput("Enter student name: ");
                std::string major = getStringInput("Enter student major: ");
                int age = getIntInput("Enter student age: ");
                manager.addStudent(name, major, age);
                break;
            }
            case 2:
                manager.displayAllStudents();
                break;
            case 3: {
                int id = getIntInput("Enter student ID: ");
                manager.displayStudent(id);
                break;
            }
            case 4: {
                int id = getIntInput("Enter student ID to update: ");
                std::string newName = getStringInput("Enter new name: ");
                std::string newMajor = getStringInput("Enter new major: ");
                int newAge = getIntInput("Enter new age: ");
                manager.updateStudent(id, newName, newMajor, newAge);
                break;
            }
            case 5: {
                int id = getIntInput("Enter student ID to delete: ");
                manager.deleteStudent(id);
                break;
            }
            case 6:
                std::cout << "Exiting program...\n";
                break;
            default:
                std::cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

✓ ရှင်းလင်းချက်

1. Student Class

- id: ကျောင်းသား၏ unique identifier
- name: ကျောင်းသားအမည်
- major: ကျောင်းသား၏ဘာသာရပ်
- age: ကျောင်းသား၏အသက်
- display(): ကျောင်းသားအချက်အလက်များကိုဖော်ပြရန်

2. StudentManager Class

- students: ကျောင်းသားများကိုသိမ်းဆည်းထားသော vector
- nextId: အသစ်ထပ်မံထည့်မည့်ကျောင်းသား၏ ID

✓ CRUD Operations:

1. Create (Add)
   - addStudent(): ကျောင်းသားအသစ်ထည့်ရန်
2. Read (View)
   - displayAllStudents(): ကျောင်းသားအားလုံးကိုကြည့်ရန်
   - displayStudent(): Specific ကျောင်းသားတစ်ဦးကိုကြည့်ရန်
3. Update
   - updateStudent(): ရှိပြီးသောကျောင်းသားအချက်အလက်ပြင်ဆင်ရန်
4. Delete

- deleteStudent(): ကျောင်းသားတစ်ဦးကိုဖျက်ရန်

3. Helper Functions

- findStudent(): ID အလိုက်ကျောင်းသားရှာရန်
- getStringInput(): String input ရယူရန်
- getIntInput(): Number input ရယူရန်

✓ အသုံးပြုပုံ အဆင့်ဆင့်

1. ကျောင်းသားအသစ်ထည့်ခြင်း

```
=== Student Management System ===
1. Add New Student
2. View All Students
3. View Student by ID
4. Update Student
5. Delete Student
6. Exit
Enter your choice: 1
Enter student name: မောင်မောင်
Enter student major: Computer Science
Enter student age: 20
Student added successfully!
```

2. ကျောင်းသားအားလုံးကြည့်ခြင်း

```
=== Student List ===

ID: 1
Name: မောင်မောင်
Major: Computer Science
Age: 20
```

3. ကျောင်းသားတစ်ဦးတည်းကြည့်ခြင်း

```
Enter student ID: 1

ID: 1
Name: မောင်မောင်
Major: Computer Science
Age: 20
```

4. ကျောင်းသားအချက်အလက်ပြင်ဆင်ခြင်း

```
Enter student ID to update: 1
Enter new name: မောင်မြတ်
Enter new major: Software Engineering
Enter new age: 21
Student updated successfully!
```

5. ကျောင်းသားဖျက်ခြင်း

```
Enter student ID to delete: 1
Student deleted successfully!
```

✓ Short Notes:

1. လွယ်ကူသော CRUD Operations - Create, Read, Update, Delete လုပ်ဆောင်ချက်များ
2. ID အလိုက် စီမံခန့်ခွဲနိုင်မှု - ကျောင်းသားတစ်ဦးချင်းစီကို ID ဖြင့်အတိအကျစီမံနိုင်သည်
3. User-Friendly Interface - အသုံးပြုရလွယ်ကူသော menu system
4. Input Validation - Number input များကိုအလိုအလျောက်စစ်ဆေးပေးသည်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

List

❒ List

C++ မှာ list ဆိုတာ Standard Template Library (STL) ရဲ့ sequence container တစ်ခုဖြစ်ပြီး doubly-linked list အဖြစ် implement လုပ်ထားပါတယ်။ List က data တွေကို non-contiguous memory locations တွေမှာ သိမ်းဆည်းပေးပါတယ်။

✓ List ၏ အခြေခံအသုံးပြုနည်း

1. List ကို Initialize လုပ်ခြင်း

```
#include <list>
#include <iostream>

int main() {
    // ဗလာ list တစ်ခု ဖန်တီးခြင်း
    std::list<int> numbers1;

    // Initializer list သုံးခြင်း
    std::list<int> numbers2 = {1, 2, 3, 4, 5};

    // Size နဲ့ initial value သတ်မှတ်ခြင်း
    std::list<int> numbers3(5, 10);  // {10, 10, 10, 10, 10}

    // Array တစ်ခုကနေ copy လုပ်ခြင်း
    int arr[] = {6, 7, 8};
    std::list<int> numbers4(arr, arr + 3);  // {6, 7, 8}

    return 0;
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

2. List ထဲကို Element တွေ ထည့်ခြင်း

```
std::list<std::string> names;

// နောက်ဆုံးမှာ element ထည့်ခြင်း
names.push_back("Alice");
names.push_back("Bob");

// ရှေ့မှာ element ထည့်ခြင်း
names.push_front("Eve");

// Specific position မှာ ထည့်ခြင်း
auto it = names.begin();
std::advance(it, 1);  // Iterator ကို position 1 ကို ရွှေ့မယ်
names.insert(it, "Charlie");  // {"Eve", "Charlie", "Alice", "Bob"}
```

3. List ထဲက Element တွေကို Access လုပ်ခြင်း

```
std::list<double> prices = {10.5, 20.3, 30.2};

// Front နဲ့ back elements
double front = prices.front();  // 10.5
double back = prices.back();    // 30.2

// Iterator သုံးပြီး access လုပ်ခြင်း
for (auto it = prices.begin(); it != prices.end(); ++it) {
    std::cout << *it << ' ';
}

// Range-based for loop (C++11)
for (double price : prices) {
    std::cout << price << ' ';
}
```

✓ List ၏ အရေးကြီးသော Member Functions

1. Size နဲ့ Capacity ဆိုင်ရာ Functions

```
std::list<int> nums = {1, 2, 3};

// List ရဲ့current size
std::cout << "Size: " << nums.size() << '\n';  // 3

// List ရဲ့maximum possible size
std::cout << "Max size: " << nums.max_size() << '\n';

// List က ဗလာလား စစ်ခြင်း
if (nums.empty()) {
    std::cout << "List is empty\n";
}
```

2. Element တွေကို ဖျက်ခြင်း

```
std::list<char> letters = {'a', 'b', 'c', 'd', 'e'};

// နောက်ဆုံး element ကို ဖျက်ခြင်း
letters.pop_back();  // {'a', 'b', 'c', 'd'}

// ရှေ့ဆုံး element ကို ဖျက်ခြင်း
letters.pop_front();  // {'b', 'c', 'd'}

// Specific element ကို ဖျက်ခြင်း
auto it = letters.begin();
std::advance(it, 1);  // 'c' ကို ရွေးမယ်
letters.erase(it);     // {'b', 'd'}

// Range of elements တွေကို ဖျက်ခြင်း
letters.erase(letters.begin(), letters.end());  // List က ဗလာဖြစ်သွားမယ်
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

3. List Operations

```
std::list<int> list1 = {1, 2, 3};
std::list<int> list2 = {4, 5, 6};

// List တွေကို ပေါင်းခြင်း
list1.splice(list1.end(), list2);  // list1 = {1,2,3,4,5,6}, list2 = {}

// List တွေကို ပြန်ခွဲခြင်း
list2.splice(list2.begin(), list1, std::next(list1.begin(), 3), list1.end());
// list1 = {1,2,3}, list2 = {4,5,6}

// List ကို sort လုပ်ခြင်း
list1 = {3, 1, 4};
list1.sort();  // {1, 3, 4}

// List ကို reverse လုပ်ခြင်း
list1.reverse();  // {4, 3, 1}

// Unique elements ပဲထားခြင်း (sort လုပ်ပြီးမှ အလုပ်လုပ်မယ်)
list1 = {1, 2, 2, 3, 3, 3};
list1.unique();  // {1, 2, 3}
```

✓ List ၏ အထူးခြားသော Features

1. Efficient Insertion/Deletion

```
std::list<int> nums = {10, 20, 30, 40};

// ဘယ် position မှာမဆို O(1) complexity နဲ့ insert/delete လုပ်နိုင်တယ်
auto it = nums.begin();
std::advance(it, 2);  // Position 2 (30)
nums.insert(it, 25);   // {10, 20, 25, 30, 40}

nums.erase(it);        // {10, 20, 25, 40} (it က 30 ကို ညွှန်းနေတယ်)
```

2. No Random Access

```
std::list<int> nums = {1, 2, 3, 4, 5};

// List မှာ random access operator [] မရှိဘူး
// nums[2] = 10;  // Error

// Iterator သုံးပြီး access လုပ်ရမယ်
auto it = nums.begin();
std::advance(it, 2);  // Position 2 (3)
*it = 10;             // {1, 2, 10, 4, 5}
```

3. Merge Operation

```
std::list<int> list1 = {1, 3, 5};
std::list<int> list2 = {2, 4, 6};

// List နှစ်ခုကို merge လုပ်ခြင်း (နှစ်ခုလုံး sort လုပ်ထားရမယ်)
list1.sort();
list2.sort();
list1.merge(list2);  // list1 = {1,2,3,4,5,6}, list2 = {}
```

List ကို အသုံးပြုရမယ့် အခြေအနေများ

1. Frequent insert/delete operations - ဘယ် position မှာမဆို O(1) complexity
2. No random access needed - Index နဲ့ တိုက်ရိုက် access မလိုရင်
3. Need to maintain order - Sort, merge, unique လိုအပ်ရင်
4. Large data sets with many modifications - Memory reallocation မရှိဘူး

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

List မသုံးသင့်တဲ့ အခြေအနေများ

1. Random access လိုအပ်တဲ့အခါ - vector သို့ array က ပိုသင့်တော်တယ်
2. Memory efficiency လိုအပ်တဲ့အခါ - List က extra memory (pointers) ယူတယ်
3. Cache performance အရေးကြီးတဲ့အခါ - Contiguous memory မဟုတ်လို့ cache miss များတယ်

Performance Comparison

Operation	list	vector	deque
Insert at begin	O(1)	O(n)	O(1)
Insert at end	O(1)	O(1)*	O(1)
Insert at middle	O(1)	O(n)	O(n)
Random access	O(n)	O(1)	O(1)
Memory usage	High	Low	Medium

*vector: O(1) amortized (reallocation လုပ်ရင် O(n))

C++ list ကို ကောင်းစွာအသုံးပြုနိုင်ရင် သင့် program တွေရဲ့ performance ကို သိသိသာသာ တိုးတက်စေနိုင်ပါတယ်။ List ရဲ့အားသာချက်တွေကို နားလည်ပြီး သင့်တော်တဲ့ အခြေအနေတွေမှာ အသုံးပြုသင့်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန်

❒ Posts CRUD Application using List

```
#include <iostream>
#include <list>
#include <string>
#include <algorithm>

class Post {
private:
    int id;
    std::string title;
    std::string content;

public:
    Post(int id, std::string title, std::string content)
        : id(id), title(title), content(content) {}

    // Getters
    int getId() const { return id; }
    std::string getTitle() const { return title; }
    std::string getContent() const { return content; }

    // Setters
    void setTitle(std::string newTitle) { title = newTitle; }
    void setContent(std::string newContent) { content = newContent; }

    // Display post
    void display() const {
        std::cout << "\nID: " << id
                  << "\nTitle: " << title
                  << "\nContent: " << content << "\n";
    }
};

class PostManager {
private:
```

```
    std::list<Post> posts;
    int nextId = 1;

public:
    // Create
    void addPost(std::string title, std::string content) {
        posts.emplace_back(nextId++, title, content);
        std::cout << "Post added successfully!\n";
    }

    // Read All
    void displayAllPosts() const {
        if (posts.empty()) {
            std::cout << "No posts available.\n";
            return;
        }
        std::cout << "\n=== All Posts ===\n";
        for (const auto& post : posts) {
            post.display();
        }
    }

    // Read Single
    void displayPost(int id) const {
        auto it = findPost(id);
        if (it != posts.end()) {
            it->display();
        } else {
            std::cout << "Post with ID " << id << " not found.\n";
        }
    }

    // Update
    void updatePost(int id, std::string newTitle, std::string newContent) {
        auto it = findPost(id);
        if (it != posts.end()) {
            it->setTitle(newTitle);
            it->setContent(newContent);
            std::cout << "Post updated successfully!\n";
        } else {
            std::cout << "Post with ID " << id << " not found.\n";
        }
```

```
    }

    // Delete
    void deletePost(int id) {
        auto it = findPost(id);
        if (it != posts.end()) {
            posts.erase(it);
            std::cout << "Post deleted successfully!\n";
        } else {
            std::cout << "Post with ID " << id << " not found.\n";
        }
    }

private:
    // Helper function to find post by ID
    std::list<Post>::iterator findPost(int id) {
        return std::find_if(posts.begin(), posts.end(),
            [id](const Post& p) { return p.getId() == id; });
    }

    // Const version for const methods
    std::list<Post>::const_iterator findPost(int id) const {
        return std::find_if(posts.begin(), posts.end(),
            [id](const Post& p) { return p.getId() == id; });
    }
};

// User interface functions
void displayMenu() {
    std::cout << "\n=== Post Management System ===\n"
              << "1. Add New Post\n"
              << "2. View All Posts\n"
              << "3. View Single Post\n"
              << "4. Update Post\n"
              << "5. Delete Post\n"
              << "6. Exit\n"
              << "Enter your choice: ";
}

std::string getInput(const std::string& prompt) {
    std::string input;
    std::cout << prompt;
```

```
    std::getline(std::cin, input);
    return input;
}

int getIntInput(const std::string& prompt) {
    int value;
    std::cout << prompt;
    while (!(std::cin >> value)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "Invalid input. Please enter a number: ";
    }
    std::cin.ignore();
    return value;
}

int main() {
    PostManager manager;
    int choice;

    do {
        displayMenu();
        choice = getIntInput("");

        switch (choice) {
            case 1: {
                std::string title = getInput("Enter post title: ");
                std::string content = getInput("Enter post content: ");
                manager.addPost(title, content);
                break;
            }
            case 2:
                manager.displayAllPosts();
                break;
            case 3: {
                int id = getIntInput("Enter post ID to view: ");
                manager.displayPost(id);
                break;
            }
            case 4: {
                int id = getIntInput("Enter post ID to update: ");
                std::string newTitle = getInput("Enter new title: ");
```

```
                std::string newContent = getInput("Enter new content: ");
                manager.updatePost(id, newTitle, newContent);
                break;
            }
            case 5: {
                int id = getIntInput("Enter post ID to delete: ");
                manager.deletePost(id);
                break;
            }
            case 6:
                std::cout << "Exiting program...\n";
                break;
            default:
                std::cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

✓ အသေးစိတ်ရှင်းလင်းချက်

1. Post Class

- id: Post ၏ unique identifier
- title: Post ခေါင်းစဉ်
- content: Post အကြောင်းအရာ
- display(): Post အချက်အလက်များကို ဖော်ပြရန်

2. PostManager Class

- posts: Post များကို သိမ်းဆည်းထားသော list
- nextId: အသစ်ထပ်မံထည့်မည့် post ၏ ID

✓ CRUD Operations:

1. Create (Add)
   - addPost(): အသစ် post တစ်ခုထည့်ရန်

2. Read (View)
   - displayAllPosts(): Post အားလုံးကိုကြည့်ရန်
   - displayPost(): Specific post တစ်ခုကိုကြည့်ရန်

3. Update
   - updatePost(): ရှိပြီးသော post ကိုပြင်ဆင်ရန်

4. Delete
   - deletePost(): Post တစ်ခုကိုဖျက်ရန်

3. Helper Functions

- findPost(): ID အလိုက် post ရှာရန်
- getInput(): User input ရယူရန်
- getIntInput(): Number input ရယူရန်

✓ အသုံးပြုပုံ အဆင့်ဆင့်

1. Post အသစ်ထည့်ခြင်း

```
=== Post Management System ===
1. Add New Post
2. View All Posts
3. View Single Post
4. Update Post
5. Delete Post
6. Exit
Enter your choice: 1
Enter post title: C++ Learning
Enter post content: Today I learned about STL containers
Post added successfully!
```

2. Post အားလုံးကြည့်ခြင်း

```
=== All Posts ===

ID: 1
Title: C++ Learning
Content: Today I learned about STL containers
```

3. Post တစ်ခုတည်းကြည့်ခြင်း

```
Enter post ID to view: 1

ID: 1
Title: C++ Learning
Content: Today I learned about STL containers
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

4. Post ပြင်ဆင်ခြင်း

```
Enter post ID to update: 1
Enter new title: C++ STL Learning
Enter new content: Today I explored list container in STL
Post updated successfully!
```

5. Post ဖျက်ခြင်း

```
Enter post ID to delete: 1
Post deleted successfully!
```

Set

❒ Set

C++ မှာ set ဆိုတာ Standard Template Library (STL) ရဲ့ associative container တစ်ခုဖြစ်ပြီး unique elements တွေကို sorted order အလိုက် သိမ်းဆည်းပေးပါတယ်။ Set က Red-Black Tree (self-balancing binary search tree) အပေါ်မှာ implement လုပ်ထားပါတယ်။

✓ Set ၏ အခြေခံအသုံးပြုနည်း

1. Set ကို Initialize လုပ်ခြင်း

```
#include <set>
#include <iostream>

int main() {
    // ဗလာ set တစ်ခု ဖန်တီးခြင်း
    std::set<int> numbers1;

    // Initializer list သုံးခြင်း
    std::set<int> numbers2 = {3, 1, 4, 1, 5};  // {1, 3, 4, 5} - duplicate ဖယ်ထုတ်မယ်

    // Array တစ်ခုကနေ copy လုပ်ခြင်း
    int arr[] = {6, 7, 8, 6};
    std::set<int> numbers3(arr, arr + 4);  // {6, 7, 8}

    return 0;
}
```

2. Set ထဲကို Element တွေ ထည့်ခြင်း

```
std::set<std::string> names;

// Insert လုပ်ခြင်း
names.insert("Alice");
names.insert("Bob");
names.insert("Alice");  // Duplicate ဖြစ်လို့ ထပ်မထည့်ဘူး


// Insert with hint
auto hint = names.begin();
names.insert(hint, "Charlie");  // Insertion position ကို hint ပေးနိုင်တယ်
```

3. Set ထဲက Element တွေကို Access လုပ်ခြင်း

```
std::set<double> prices = {10.5, 20.3, 30.2};

// Iterator သုံးပြီး access လုပ်ခြင်း
for (auto it = prices.begin(); it != prices.end(); ++it) {
    std::cout << *it << ' ';
}

// Range-based for loop (C++11)
for (double price : prices) {
    std::cout << price << ' ';
}

// Find သုံးပြီး access လုပ်ခြင်း
auto it = prices.find(20.3);
if (it != prices.end()) {
    std::cout << "Found: " << *it << '\n';
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ Set ၏ အရေးကြီးသော Member Functions

1. Size နဲ့ Capacity ဆိုင်ရာ Functions

```
std::set<int> nums = {1, 2, 3};

// Set ရဲ့current size
std::cout << "Size: " << nums.size() << '\n';  // 3

// Set ရဲ့maximum possible size
std::cout << "Max size: " << nums.max_size() << '\n';

// Set က ဗလာလား စစ်ခြင်း
if (nums.empty()) {
    std::cout << "Set is empty\n";
}
```

2. Element တွေကို ဖျက်ခြင်း

```
std::set<char> letters = {'a', 'b', 'c', 'd', 'e'};

// Specific element ကို ဖျက်ခြင်း
letters.erase('b');  // {'a', 'c', 'd', 'e'}

// Iterator သုံးပြီး ဖျက်ခြင်း
auto it = letters.find('d');
if (it != letters.end()) {
    letters.erase(it);  // {'a', 'c', 'e'}
}

// Range of elements တွေကို ဖျက်ခြင်း
letters.erase(letters.begin(), letters.end());  // Set က ဗလာဖြစ်သွားမယ်
```

3. Search Operations

```
std::set<int> numbers = {10, 20, 30, 40, 50};

// Count သုံးပြီး element ရှိမရှိစစ်ခြင်း (set မှာ 0 သို့ 1 ဘဲပြန်မယ်)
if (numbers.count(30) > 0) {
    std::cout << "30 found in set\n";
}

// Lower bound (ပေးထားတဲ့ value နဲ့ညီမျှသို့မဟုတ် ကြီးတဲ့ first element)
auto lb = numbers.lower_bound(25);  // 30 ကိုညွှန်းမယ်

// Upper bound (ပေးထားတဲ့ value ထက် ကြီးတဲ့ first element)
auto ub = numbers.upper_bound(35);  // 40 ကိုညွှန်းမယ်

// Equal range (lower bound နဲ့ upper bound pair ပြန်ပေးမယ်)
auto range = numbers.equal_range(30);
```

✓ Set ၏ အထူးခြားသော Features

1. Automatic Sorting

```
std::set<int> nums = {5, 3, 8, 1, 2};

// Set က automatically sort လုပ်ပေးမယ်
for (int n : nums) {
    std::cout << n << ' ';  // 1 2 3 5 8
}
```

2. Unique Elements

```
std::set<int> numbers;
numbers.insert(10);
numbers.insert(20);
numbers.insert(10);  // Duplicate ဖြစ်လို့ မထည့်ဘူး

std::cout << numbers.size();  // 2
```

3. Custom Comparator

```
// Descending order အတွက် custom comparator
struct Greater {
    bool operator()(int a, int b) const {
        return a > b;
    }
};

std::set<int, Greater> descending = {3, 1, 4, 2};

for (int n : descending) {
    std::cout << n << ' ';  // 4 3 2 1
}
```

✓ Set ကို အသုံးပြုရမယ့် အခြေအနေများ

1. Unique elements လိုအပ်တဲ့အခါ - Duplicate တွေကို အလိုအလျောက် ဖယ်ရှားပေးမယ်
2. Sorted order လိုအပ်တဲ့အခါ - Element တွေကို အလိုအလျောက် sort လုပ်ပေးမယ်
3. Fast search လိုအပ်တဲ့အခါ - O(log n) complexity နဲ့ search လုပ်နိုင်တယ်
4. Range queries လိုအပ်တဲ့အခါ - lower_bound, upper_bound တွေနဲ့ range query လုပ်နိုင်တယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ Set မသုံးသင့်တဲ့ အခြေအနေများ

1. Duplicate values လိုအပ်တဲ့အခါ - multiset ကို သုံးသင့်တယ်
2. Frequent modifications လုပ်ရတဲ့အခါ - Insert/delete က O(log n) complexity ရှိတယ်
3. Memory efficiency အရေးကြီးတဲ့အခါ - Set က extra memory (tree structure) ယူတယ်

Performance Comparison

Operation	set	unordered_set	vector
Insert	O(log n)	O(1) avg	O(1) (end), O(n) (middle)
Delete	O(log n)	O(1) avg	O(n)
Search	O(log n)	O(1) avg	O(n)
Memory usage	High	Medium	Low
Order maintained	Yes	No	Yes

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန်

❒ Car Sale CRUD Application using Set

```
#include <iostream>
#include <set>
#include <string>
#include <algorithm>

class Car {
private:
    std::string licensePlate;  // Unique identifier
    std::string brand;
    std::string model;
    double price;

public:
    Car(std::string plate, std::string brand, std::string model, double price
)
        : licensePlate(plate), brand(brand), model(model), price(price) {}

    // Getters
    std::string getLicensePlate() const { return licensePlate; }
    std::string getBrand() const { return brand; }
    std::string getModel() const { return model; }
    double getPrice() const { return price; }

    // For set ordering
    bool operator<(const Car& other) const {
        return licensePlate < other.licensePlate;
    }

    // Display car information
    void display() const {
        std::cout << "\nLicense Plate: " << licensePlate
                  << "\nBrand: " << brand
                  << "\nModel: " << model
                  << "\nPrice: $" << price << "\n";
    }
};
```

```
class CarShowroom {
private:
    std::set<Car> cars;

public:
    // Add a new car
    void addCar(std::string plate, std::string brand, std::string model, doub
le price) {
        auto result = cars.emplace(plate, brand, model, price);
        if (result.second) {
            std::cout << "Car added successfully!\n";
        } else {
            std::cout << "Car with license plate " << plate << " already exis
ts!\n";
        }
    }

    // View all cars
    void displayAllCars() const {
        if (cars.empty()) {
            std::cout << "No cars available in showroom.\n";
            return;
        }

        std::cout << "\n=== Cars in Showroom ===\n";
        for (const auto& car : cars) {
            car.display();
        }
    }

    // View single car
    void displayCar(std::string plate) const {
        auto it = cars.find(Car(plate, "", "", 0));
        if (it != cars.end()) {
            it->display();
        } else {
            std::cout << "Car with license plate " << plate << " not found.\n
";
        }
    }
```

```
    // Update car price
    void updateCarPrice(std::string plate, double newPrice) {
        auto it = cars.find(Car(plate, "", "", 0));
        if (it != cars.end()) {
            // Create new car with updated price
            Car updatedCar = *it;
            updatedCar = Car(plate, it->getBrand(), it->getModel(), newPrice)
;

            // Remove old and add new
            cars.erase(it);
            cars.insert(updatedCar);

            std::cout << "Car price updated successfully!\n";
        } else {
            std::cout << "Car with license plate " << plate << " not found.\n
";
        }
    }

    // Remove a car
    void removeCar(std::string plate) {
        auto it = cars.find(Car(plate, "", "", 0));
        if (it != cars.end()) {
            cars.erase(it);
            std::cout << "Car removed successfully!\n";
        } else {
            std::cout << "Car with license plate " << plate << " not found.\n
";
        }
    }
};

// User interface functions
void displayMenu() {
    std::cout << "\n=== Car Showroom Management ===\n"
              << "1. Add New Car\n"
              << "2. View All Cars\n"
              << "3. View Car by License Plate\n"
              << "4. Update Car Price\n"
              << "5. Remove Car\n"
              << "6. Exit\n"
```

```
              << "Enter your choice: ";
}

std::string getStringInput(const std::string& prompt) {
    std::string input;
    std::cout << prompt;
    std::getline(std::cin >> std::ws, input);
    return input;
}

double getDoubleInput(const std::string& prompt) {
    double value;
    std::cout << prompt;
    while (!(std::cin >> value)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "Invalid input. Please enter a number: ";
    }
    std::cin.ignore();
    return value;
}

int main() {
    CarShowroom showroom;
    int choice;

    do {
        displayMenu();
        choice = getDoubleInput("");

        switch (choice) {
            case 1: {
                std::string plate = getStringInput("Enter license plate: ");
                std::string brand = getStringInput("Enter car brand: ");
                std::string model = getStringInput("Enter car model: ");
                double price = getDoubleInput("Enter car price: $");
                showroom.addCar(plate, brand, model, price);
                break;
            }
            case 2:
                showroom.displayAllCars();
                break;
```

```
            case 3: {
                std::string plate = getStringInput("Enter license plate: ");
                showroom.displayCar(plate);
                break;
            }
            case 4: {
                std::string plate = getStringInput("Enter license plate: ");
                double newPrice = getDoubleInput("Enter new price: $");
                showroom.updateCarPrice(plate, newPrice);
                break;
            }
            case 5: {
                std::string plate = getStringInput("Enter license plate: ");
                showroom.removeCar(plate);
                break;
            }
            case 6:
                std::cout << "Exiting program...\n";
                break;
            default:
                std::cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

✓ အသေးစိတ်ရှင်းပြချက်

1. Car Class

- licensePlate: ကားနံပါတ် (Unique identifier)
- brand: ကားအမှတ်တံဆိပ်
- model: ကားမော်ဒယ်
- price: ကားဈေးနှုန်း
- operator<: Set အတွက် အလိုအလျောက်စီရန်

- display(): ကားအချက်အလက်များကိုဖော်ပြရန်

2. CarShowroom Class

- cars: ကားများကိုသိမ်းဆည်းထားသော set

✓ CRUD Operations:

1. Create (Add)
   - addCar(): ကားအသစ်ထည့်ရန်
2. Read (View)
   - displayAllCars(): ကားအားလုံးကိုကြည့်ရန်
   - displayCar(): Specific ကားတစ်စီးကိုကြည့်ရန်
3. Update
   - updateCarPrice(): ကားဈေးနှုန်းပြင်ဆင်ရန်
4. Delete
   - removeCar(): ကားတစ်စီးကိုဖျက်ရန်

3. Helper Functions

- getStringInput(): String input ရယူရန်
- getDoubleInput(): Number input ရယူရန်

✓ အသုံးပြုပုံ အဆင့်ဆင့်

1. ကားအသစ်ထည့်ခြင်း

```
=== Car Showroom Management ===
1. Add New Car
2. View All Cars
3. View Car by License Plate
4. Update Car Price
5. Remove Car
6. Exit
Enter your choice: 1
Enter license plate: YGN-1234
Enter car brand: Toyota
Enter car model: Corolla
Enter car price: $25000
Car added successfully!
```

2. ကားအားလုံးကြည့်ခြင်း

```
=== Cars in Showroom ===

License Plate: YGN-1234
Brand: Toyota
Model: Corolla
Price: $25000
```

3. ကားတစ်စီးတည်းကြည့်ခြင်း

```
Enter license plate: YGN-1234

License Plate: YGN-1234
Brand: Toyota
Model: Corolla
Price: $25000
```

4. ကားဈေးနှုန်းပြင်ဆင်ခြင်း

Copy
Download
```
Enter license plate: YGN-1234
Enter new price: $23000
Car price updated successfully!
```

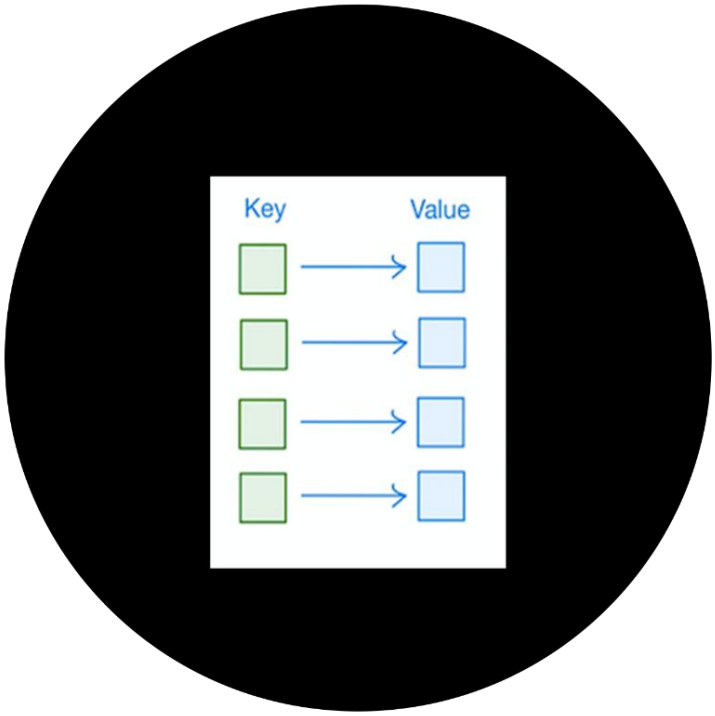
5. ကားဖျက်ခြင်း

```
Enter license plate: YGN-1234
Car removed successfully!
```

✓ Short Notes:

1. Unique License Plates - Set ကြောင့် ကားနံပါတ်တူညီသော ကားများမထည့်နိုင်
2. Automatic Sorting - ကားနံပါတ်အလိုက် အလိုအလျောက်စီပေးမည်
3. Efficient Searching - O(log n) complexity ဖြင့်ရှာဖွေနိုင်မည်
4. Clean Interface - အသုံးပြုရလွယ်ကူသော menu system

MAP



❒ Map

C++ မှာ map ဆိုတာ Standard Template Library (STL) ရဲ့ associative container တစ်ခုဖြစ်ပြီး key-value pair တွေကို sorted order (key အလိုက်) သိမ်းဆည်းပေးပါတယ်။ Map က Red-Black Tree (self-balancing binary search tree) အပေါ်မှာ implement လုပ်ထားပါတယ်။

✓ Map ၏ အခြေခံအသုံးပြုနည်း

1. Map ကို Initialize လုပ်ခြင်း

```
#include <map>
#include <iostream>

int main() {
    // ဗလာ map တစ်ခု ဖန်တီးခြင်း
    std::map<std::string, int> ageMap1;

    // Initializer list သုံးခြင်း
    std::map<std::string, int> ageMap2 = {
        {"Alice", 25},
        {"Bob", 30},
        {"Charlie", 28}
    };

    return 0;
}
```

2. Map ထဲကို Element တွေ ထည့်ခြင်း

```
std::map<std::string, double> priceMap;

// Insert လုပ်ခြင်း (pair သုံးပြီး)
priceMap.insert(std::pair<std::string, double>("Apple", 1.99));

// Insert လုပ်ခြင်း (make_pair သုံးပြီး)
priceMap.insert(std::make_pair("Banana", 0.99));

// [] operator သုံးပြီး insert/update လုပ်ခြင်း
priceMap["Orange"] = 1.49;  // Insert new
priceMap["Apple"] = 2.99;   // Update existing
```

3. Map ထဲက Element တွေကို Access လုပ်ခြင်း

```
std::map<int, std::string> idMap = {
    {101, "Alice"},
    {102, "Bob"},
    {103, "Charlie"}
};

// [] operator သုံးပြီး access လုပ်ခြင်း
std::cout << idMap[102] << '\n';  // "Bob"

// at() method သုံးပြီး access လုပ်ခြင်း (bounds checking ပါတယ်)
std::cout << idMap.at(101) << '\n';  // "Alice"

// Iterator သုံးပြီး access လုပ်ခြင်း
for (auto it = idMap.begin(); it != idMap.end(); ++it) {
    std::cout << it->first << ": " << it->second << '\n';
}

// Range-based for loop (C++11)
for (const auto& pair : idMap) {
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
    std::cout << pair.first << ": " << pair.second << '\n';
}
```

✓ Map ၏ အရေးကြီးသော Member Functions

1. Size နဲ့ Capacity ဆိုင်ရာ Functions

```
std::map<std::string, int> populationMap = {
    {"Yangon", 7000000},
    {"Mandalay", 2000000}
};

// Map ရဲ့current size
std::cout << "Size: " << populationMap.size() << '\n';  // 2

// Map ရဲ့maximum possible size
std::cout << "Max size: " << populationMap.max_size() << '\n';

// Map က ဗလာလား စစ်ခြင်း
if (populationMap.empty()) {
    std::cout << "Map is empty\n";
}
```

2. Element တွေကို ဖျက်ခြင်း

```
std::map<int, std::string> studentMap = {
    {101, "Alice"},
    {102, "Bob"},
    {103, "Charlie"}
};

// Key သုံးပြီး ဖျက်ခြင်း
studentMap.erase(102);  // {101: "Alice", 103: "Charlie"}

// Iterator သုံးပြီး ဖျက်ခြင်း
auto it = studentMap.find(101);
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
if (it != studentMap.end()) {
    studentMap.erase(it);  // {103: "Charlie"}
}

// Range of elements တွေကို ဖျက်ခြင်း
studentMap.erase(studentMap.begin(), studentMap.end());  // Map က ဗလာဖြစ်သွားမယ်
```

3. Search Operations

```
std::map<std::string, int> scoreMap = {
    {"Alice", 85},
    {"Bob", 92},
    {"Charlie", 78}
};

// Find သုံးပြီး search လုပ်ခြင်း
auto it = scoreMap.find("Bob");
if (it != scoreMap.end()) {
    std::cout << "Found: " << it->second << '\n';  // 92
}

// Count သုံးပြီး key ရှိမရှိစစ်ခြင်း (map မှာ 0 သို့ 1 ဘဲပြန်မယ်)
if (scoreMap.count("Alice") > 0) {
    std::cout << "Alice exists\n";
}

// Lower bound (ပေးထားတဲ့ key နဲ့ညီမျှသို့မဟုတ် ကြီးတဲ့ first element)
auto lb = scoreMap.lower_bound("Bob");  // "Bob" ကိုညွှန်းမယ်

// Upper bound (ပေးထားတဲ့ key ထက် ကြီးတဲ့ first element)
auto ub = scoreMap.upper_bound("Bob");  // "Charlie" ကိုညွှန်းမယ်

// Equal range (lower bound နဲ့ upper bound pair ပြန်ပေးမယ်)
auto range = scoreMap.equal_range("Alice");
```

✓ Map ၏ အထူးခြားသော Features

1. Automatic Sorting by Key

```
std::map<std::string, int> wordCount = {
    {"banana", 5},
    {"apple", 3},
    {"cherry", 7}
};

// Map က key အလိုက် automatically sort လုပ်ပေးမယ်
for (const auto& pair : wordCount) {
    std::cout << pair.first << ": " << pair.second << '\n';
}
// Output:
// apple: 3
// banana: 5
// cherry: 7
```

2. Unique Keys

```
std::map<int, std::string> idMap;
idMap[101] = "Alice";
idMap[101] = "Bob";  // Existing value ကို overwrite လုပ်မယ်

std::cout << idMap.size();  // 1
```

3. Custom Comparator

```
// Descending order အတွက် custom comparator
struct Greater {
    bool operator()(const std::string& a, const std::string& b) const {
        return a > b;
    }
};

std::map<std::string, int, Greater> descendingMap = {
```

```
    {"Apple", 10},
    {"Banana", 20},
    {"Cherry", 30}
};

for (const auto& pair : descendingMap) {
    std::cout << pair.first << ": " << pair.second << '\n';
}
// Output:
// Cherry: 30
// Banana: 20
// Apple: 10
```

Map ကို အသုံးပြုရမယ့် အခြေအနေများ

1. Key-value pairs လိုအပ်တဲ့အခါ - Data တွေကို key အလိုက် organize လုပ်ချင်ရင်
2. Sorted order လိုအပ်တဲ့အခါ - Key အလိုက် အလိုအလျောက် sort လုပ်ပေးမယ်
3. Fast lookup by key လိုအပ်တဲ့အခါ - O(log n) complexity နဲ့ search လုပ်နိုင်တယ်
4. Unique keys လိုအပ်တဲ့အခါ - တူညီတဲ့ key နှစ်ခု မရှိစေချင်ရင်

Map မသုံးသင့်တဲ့ အခြေအနေများ

1. Duplicate keys လိုအပ်တဲ့အခါ - multimap ကို သုံးသင့်တယ်
2. Unordered storage လိုအပ်တဲ့အခါ - unordered_map က ပိုမြန်တယ်
3. Memory efficiency အရေးကြီးတဲ့အခါ - Map က extra memory (tree structure) ယူတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ Performance Comparison

Operation	map	unordered_map	vector of pairs
Insert	O(log n)	O(1) avg	O(n)
Delete	O(log n)	O(1) avg	O(n)
Search by key	O(log n)	O(1) avg	O(n)
Memory usage	High	Medium	Low
Order maintained	Yes	No	Yes (if sorted)

C++ map ကို ကောင်းစွာအသုံးပြုနိုင်ရင် သင့် program တွေရဲ့ data organization နဲ့ retrieval efficiency တွေကို သိသိသာသာ တိုးတက်စေနိုင်ပါတယ်။ Map ရဲ့ sorted နဲ့ unique properties တွေက data processing တွေမှာ အထူးအသုံးဝင်ပါတယ်။

✓ လေ့ကျင့်ရန်

❒ Employees CRUD Application using Map

```
#include <iostream>
#include <map>
#include <string>
#include <limits>

class Employee {
private:
```

```
    int id;
    std::string name;
    std::string position;
    double salary;

public:
    Employee(int id = 0, std::string name = "", std::string position = "", do
uble salary = 0.0)
        : id(id), name(name), position(position), salary(salary) {}

    // Getters
    int getId() const { return id; }
    std::string getName() const { return name; }
    std::string getPosition() const { return position; }
    double getSalary() const { return salary; }

    // Setters
    void setName(std::string newName) { name = newName; }
    void setPosition(std::string newPosition) { position = newPosition; }
    void setSalary(double newSalary) { salary = newSalary; }

    // Display employee information
    void display() const {
        std::cout << "\nEmployee ID: " << id
                  << "\nName: " << name
                  << "\nPosition: " << position
                  << "\nSalary: $" << salary << "\n";
    }
};

class EmployeeManager {
private:
    std::map<int, Employee> employees;  // Key: employee ID, Value: Employee o
bject

public:
    // Create a new employee
    void addEmployee(int id, std::string name, std::string position, double s
alary) {
        auto result = employees.emplace(id, Employee(id, name, position, sala
ry));
        if (result.second) {
```

```
            std::cout << "Employee added successfully!\n";
        } else {
            std::cout << "Employee with ID " << id << " already exists!\n";
        }
    }

    // Read all employees
    void displayAllEmployees() const {
        if (employees.empty()) {
            std::cout << "No employees found.\n";
            return;
        }

        std::cout << "\n=== Employee List ===\n";
        for (const auto& pair : employees) {
            pair.second.display();
        }
    }

    // Read single employee
    void displayEmployee(int id) const {
        auto it = employees.find(id);
        if (it != employees.end()) {
            it->second.display();
        } else {
            std::cout << "Employee with ID " << id << " not found.\n";
        }
    }

    // Update employee information
    void updateEmployee(int id, std::string newName, std::string newPosition,
double newSalary) {
        auto it = employees.find(id);
        if (it != employees.end()) {
            it->second.setName(newName);
            it->second.setPosition(newPosition);
            it->second.setSalary(newSalary);
            std::cout << "Employee updated successfully!\n";
        } else {
            std::cout << "Employee with ID " << id << " not found.\n";
        }
    }
```

```
    // Delete an employee
    void deleteEmployee(int id) {
        if (employees.erase(id)) {
            std::cout << "Employee deleted successfully!\n";
        } else {
            std::cout << "Employee with ID " << id << " not found.\n";
        }
    }
};

// User interface functions
void displayMenu() {
    std::cout << "\n=== Employee Management System ===\n"
              << "1. Add New Employee\n"
              << "2. View All Employees\n"
              << "3. View Employee by ID\n"
              << "4. Update Employee\n"
              << "5. Delete Employee\n"
              << "6. Exit\n"
              << "Enter your choice: ";
}

std::string getStringInput(const std::string& prompt) {
    std::string input;
    std::cout << prompt;
    std::getline(std::cin >> std::ws, input);  // ws to skip any leftover newl
ines
    return input;
}

int getIntInput(const std::string& prompt) {
    int value;
    std::cout << prompt;
    while (!(std::cin >> value)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "Invalid input. Please enter an integer: ";
    }
    std::cin.ignore();  // Ignore leftover newline
    return value;
}
```

```
double getDoubleInput(const std::string& prompt) {
    double value;
    std::cout << prompt;
    while (!(std::cin >> value)) {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cout << "Invalid input. Please enter a number: ";
    }
    std::cin.ignore();  // Ignore leftover newline
    return value;
}

int main() {
    EmployeeManager manager;
    int choice;

    do {
        displayMenu();
        choice = getIntInput("");

        switch (choice) {
            case 1: {
                int id = getIntInput("Enter employee ID: ");
                std::string name = getStringInput("Enter employee name: ");
                std::string position = getStringInput("Enter employee positio
n: ");
                double salary = getDoubleInput("Enter employee salary: $");
                manager.addEmployee(id, name, position, salary);
                break;
            }
            case 2:
                manager.displayAllEmployees();
                break;
            case 3: {
                int id = getIntInput("Enter employee ID: ");
                manager.displayEmployee(id);
                break;
            }
            case 4: {
                int id = getIntInput("Enter employee ID to update: ");
                std::string newName = getStringInput("Enter new name: ");
```

```
                std::string newPosition = getStringInput("Enter new position:
");
                double newSalary = getDoubleInput("Enter new salary: $");
                manager.updateEmployee(id, newName, newPosition, newSalary);
                break;
            }
            case 5: {
                int id = getIntInput("Enter employee ID to delete: ");
                manager.deleteEmployee(id);
                break;
            }
            case 6:
                std::cout << "Exiting program...\n";
                break;
            default:
                std::cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

✓ အသေးစိတ်ရှင်းပြချက်

1. Employee Class

- id: ဝန်ထမ်း ID (Unique identifier)
- name: ဝန်ထမ်းအမည်
- position: ရာထူး
- salary: လစာ
- display(): ဝန်ထမ်းအချက်အလက်များကိုဖော်ပြရန်

2. EmployeeManager Class

- employees: ဝန်ထမ်းများကိုသိမ်းဆည်းထားသော map (key: ID, value: Employee object)

✓ CRUD Operations:

1. Create (Add)
    - addEmployee(): ဝန်ထမ်းအသစ်ထည့်ရန်

2. Read (View)
    - displayAllEmployees(): ဝန်ထမ်းအားလုံးကိုကြည့်ရန်
    - displayEmployee(): Specific ဝန်ထမ်းတစ်ဦးကိုကြည့်ရန်

3. Update
    - updateEmployee(): ရှိပြီးသောဝန်ထမ်းအချက်အလက်ပြင်ဆင်ရန်

4. Delete
    - deleteEmployee(): ဝန်ထမ်းတစ်ဦးကိုဖျက်ရန်

3. Helper Functions

- getStringInput(): String input ရယူရန်
- getIntInput(): Integer input ရယူရန်
- getDoubleInput(): Double input ရယူရန်

✓ အသုံးပြုပုံ အဆင့်ဆင့်

1. ဝန်ထမ်းအသစ်ထည့်ခြင်း

```
=== Employee Management System ===
1. Add New Employee
2. View All Employees
3. View Employee by ID
4. Update Employee
5. Delete Employee
6. Exit
Enter your choice: 1
Enter employee ID: 101
Enter employee name: မောင်မောင်
Enter employee position: Developer
Enter employee salary: $1200
Employee added successfully!
```

2. ဝန်ထမ်းအားလုံးကြည့်ခြင်း

```
=== Employee List ===

Employee ID: 101
Name: မောင်မောင်
Position: Developer
Salary: $1200
```

3. ဝန်ထမ်းတစ်ဦးတည်းကြည့်ခြင်း

```
Enter employee ID: 101

Employee ID: 101
Name: မောင်မောင်
Position: Developer
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
Salary: $1200
```

4. ဝန်ထမ်းအချက်အလက်ပြင်ဆင်ခြင်း

```
Enter employee ID to update: 101
Enter new name: မောင်မြတ်
Enter new position: Senior Developer
Enter new salary: $1500
Employee updated successfully!
```

5. ဝန်ထမ်းဖျက်ခြင်း

```
Enter employee ID to delete: 101
Employee deleted successfully!
```

✓ Short Notes:

1. ID အလိုက် အလိုအလျောက်စီခြင်း - Map ကြောင့် ဝန်ထမ်း ID များကို အလိုအလျောက်စီပေးမည်
2. အချိန်တိုအတွင်း ရှာဖွေနိုင်ခြင်း - O(log n) complexity ဖြင့် ဝန်ထမ်းများကို ID ဖြင့်လွယ်ကူစွာရှာဖွေနိုင်မည်
3. Unique IDs - တူညီသော ID နှစ်ခုမရှိစေရန် အလိုအလျောက်စစ်ဆေးပေးမည်
4. လွယ်ကူသော Update/Delete - ID ဖြင့် တိုက်ရိုက်အချက်အလက်ပြင်ဆင်နိုင်ခြင်း

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Sorting Algorithms

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ Sorting Algorithms

C++ မှာ အသုံးများတဲ့ sorting algorithms များနှင့် ၎င်းတို့ကို အသုံးပြုနည်းများကို ရှင်းပြပေးပါမယ်။

1. Standard Library Sort (std::sort)

C++ Standard Template Library (STL) မှာ ပါဝင်တဲ့ အထိရောက်ဆုံး sorting algorithm ဖြစ်ပါတယ်။

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> numbers = {5, 2, 9, 1, 5, 6};

    // Ascending order (တိုးကာရီ)
    std::sort(numbers.begin(), numbers.end());

    std::cout << "Ascending order: ";
    for(int num : numbers) {
        std::cout << num << " ";
    }

    // Descending order (လျော့ကာရီ)
    std::sort(numbers.begin(), numbers.end(), std::greater<int>());

    std::cout << "\nDescending order: ";
    for(int num : numbers) {
        std::cout << num << " ";
    }
```

```
    return 0;
}
```

2. Bubble Sort (အလွယ်ဆုံး Sorting Algorithm)

```
void bubbleSort(int arr[], int n) {
    for(int i = 0; i < n-1; i++) {
        for(int j = 0; j < n-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                // Swap elements
                std::swap(arr[j], arr[j+1]);
            }
        }
    }
}

// အသုံးပြုနည်း:
int arr[] = {64, 34, 25, 12, 22, 11, 90};
int n = sizeof(arr)/sizeof(arr[0]);
bubbleSort(arr, n);
```

3. Selection Sort

```
void selectionSort(int arr[], int n) {
    for(int i = 0; i < n-1; i++) {
        int min_idx = i;
        for(int j = i+1; j < n; j++) {
            if(arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        std::swap(arr[min_idx], arr[i]);
    }
}
```

4. Insertion Sort

```
void insertionSort(int arr[], int n) {
    for(int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while(j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
}
```

5. Quick Sort (အမြန်ဆုံး Algorithms တစ်ခု)

```
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for(int j = low; j <= high-1; j++) {
        if(arr[j] < pivot) {
            i++;
            std::swap(arr[i], arr[j]);
        }
    }
    std::swap(arr[i+1], arr[high]);
    return (i+1);
}

void quickSort(int arr[], int low, int high) {
    if(low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi-1);
        quickSort(arr, pi+1, high);
    }
```

```
}

// အသုံးပြုနည်း:
int arr[] = {10, 7, 8, 9, 1, 5};
int n = sizeof(arr)/sizeof(arr[0]);
quickSort(arr, 0, n-1);
```

6. Merge Sort

```
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for(int i = 0; i < n1; i++)
        L[i] = arr[l+i];
    for(int j = 0; j < n2; j++)
        R[j] = arr[m+1+j];

    int i = 0, j = 0, k = l;

    while(i < n1 && j < n2) {
        if(L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while(i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
```

```
    while(j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if(l < r) {
        int m = l + (r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

✓ Algorithm များ၏ အချိန်ရှုပ်ထွေးမှု (Time Complexity)

Algorithm	Best Case	Average Case	Worst Case	Space Complexity
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)
std::sort	O(n log n)	O(n log n)	O(n log n)	O(log n)

✓ Custom Object Sorting

```
#include <algorithm>
#include <vector>

class Person {
public:
    std::string name;
    int age;

    Person(std::string n, int a) : name(n), age(a) {}
};

bool compareByAge(const Person &a, const Person &b) {
    return a.age < b.age;
}

int main() {
    std::vector<Person> people = {
        Person("Alice", 25),
        Person("Bob", 20),
        Person("Charlie", 30)
    };

    // Age အလိုက် စီခြင်း
    std::sort(people.begin(), people.end(), compareByAge);

    // Lambda function ဖြင့် စီခြင်း
    std::sort(people.begin(), people.end(), [](const Person &a, const Person
&b) {
        return a.name < b.name;
    });

    return 0;
}
```

✓ ဘယ် Algorithm ကို ရွေးချယ်သုံးစွဲမလဲ?

1. အသေးစား dataset များအတွက် - Insertion Sort
2. အလယ်အလတ် dataset များအတွက် - std::sort (IntroSort)
3. အကြီးစား dataset များအတွက် - Merge Sort or Quick Sort
4. Memory ကန့်သတ်ချက်ရှိလျှင် - Heap Sort
5. Stability လိုအပ်လျှင် - Merge Sort

မှတ်ချက်: C++ မှာ std::sort က အကောင်းဆုံးရွေးချယ်မှုဖြစ်ပြီး hybrid algorithm (IntroSort) ကိုသုံးထားတာကြောင့် အများအားဖြင့် အကောင်းဆုံးစွမ်းဆောင်ရည်ကို ပေးစွမ်းနိုင်ပါတယ်။

✓ လေ့ကျင့်ရန်

❒ C++ Console Stock Item Management Application

⇨ အခြေခံအချက်အလက်များ

- ဒီ application မှာ stock item များကို စီမံခန့်ခွဲနိုင်မယ့် console program တစ်ခုဖြစ်ပါတယ်
- Menu 3 ခုပါဝင်မယ်:
    1. Stock List အားလုံးကို ပြမယ်
    2. Category အလိုက် စီပြမယ်
    3. Price အလိုက် စီပြမယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

⇨ အပြည့်အစုံ Program Code

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <string>
using namespace std;

// Stock Item Class
class StockItem {
public:
    string id;
    string name;
    string category;
    double price;
    int quantity;

    StockItem(string i, string n, string c, double p, int q)
        : id(i), name(n), category(c), price(p), quantity(q) {}
};

// Stock Database
vector<StockItem> stockDatabase = {
    StockItem("1001", "Laptop", "Electronics", 1200.99, 15),
    StockItem("1002", "Smartphone", "Electronics", 899.50, 30),
    StockItem("2001", "T-Shirt", "Clothing", 19.99, 50),
    StockItem("2002", "Jeans", "Clothing", 49.99, 25),
    StockItem("3001", "Coffee", "Groceries", 8.99, 100),
    StockItem("3002", "Tea", "Groceries", 5.99, 80),
    StockItem("4001", "Hammer", "Tools", 12.50, 40),
    StockItem("4002", "Screwdriver", "Tools", 7.99, 60)
};

// Function Prototypes
void displayMenu();
void showStockList();
void sortByCategory();
```

```
void sortByPrice();
void displayStockTable(const vector<StockItem>& items);

int main() {
    int choice;

    do {
        displayMenu();
        cout << "လုပ်ဆောင်ချက် ရွေးချယ်ပါ (1-4): ";
        cin >> choice;

        switch(choice) {
            case 1:
                showStockList();
                break;
            case 2:
                sortByCategory();
                break;
            case 3:
                sortByPrice();
                break;
            case 4:
                cout << "Program မှ ထွက်ခွာပါမည်..." << endl;
                break;
            default:
                cout << "မှားယွင်းသော ရွေးချယ်မှု! 1 မှ 4 အထိ နံပါတ်များသာ ရွေးချယ်ပါ။" << endl
;
        }

        if(choice != 4) {
            cout << "\nEnter နှိပ်ပါ ဆက်လက်ဆောင်ရွက်ရန်...";
            cin.ignore();
            cin.get();
        }

    } while(choice != 4);

    return 0;
}

// Menu Display Function
```

```
void displayMenu() {
    system("cls"); // Windows - console clear
    // system("clear"); // Linux/macOS

    cout << "====================================" << endl;
    cout << "    Stock Item Management System" << endl;
    cout << "====================================" << endl;
    cout << "1. Stock List အားလုံးကိုကြည့်မည်" << endl;
    cout << "2. Category အလိုက်စီပြမည်" << endl;
    cout << "3. Price အလိုက်စီပြမည်" << endl;
    cout << "4. Program မှ ထွက်ခွာမည်" << endl;
    cout << "====================================" << endl;
}

// Show All Stock Items
void showStockList() {
    displayStockTable(stockDatabase);
}

// Sort by Category
void sortByCategory() {
    vector<StockItem> sortedItems = stockDatabase;

    // Category အလိုက်စီပြီး Name အလိုက်စီမည်
    sort(sortedItems.begin(), sortedItems.end(), [](const StockItem& a, const
StockItem& b) {
        if(a.category != b.category) {
            return a.category < b.category;
        }
        return a.name < b.name;
    });

    displayStockTable(sortedItems);
}

// Sort by Price
void sortByPrice() {
    vector<StockItem> sortedItems = stockDatabase;

    // Price အလိုက်စီမည် (စျေးနည်းရာမှ အများဆီ)
```

```
    sort(sortedItems.begin(), sortedItems.end(), [](const StockItem& a, const
StockItem& b) {
        return a.price < b.price;
    });

    displayStockTable(sortedItems);
}

// Display Stock Items in Table Format
void displayStockTable(const vector<StockItem>& items) {
    cout << "\nStock Item List:\n";
    cout << "-----------------------------------------------------------------
----\n";
    cout << left << setw(10) << "ID" << setw(20) << "Name"
         << setw(15) << "Category" << setw(10) << "Price" << setw(10) << "Qty
\n";
    cout << "-----------------------------------------------------------------
----\n";

    for(const auto& item : items) {
        cout << left << setw(10) << item.id
             << setw(20) << item.name
             << setw(15) << item.category
             << "$" << setw(9) << fixed << setprecision(2) << item.price
             << setw(10) << item.quantity << endl;
    }

    cout << "-----------------------------------------------------------------
----\n";
    cout << "စုစုပေါင်း " << items.size() << " မျိုးတွေ့ရှိပါသည်။\n";
}
```

✓ Program အလုပ်လုပ်ပုံ အဆင့်ဆင့်

1. အချက်အလက်များ သတ်မှတ်ခြင်း
    - StockItem class ဖြင့် ပစ္စည်းတစ်ခုစီရဲ့အချက်အလက်များကို သိမ်းဆည်းပါတယ်
    - stockDatabase vector ထဲတွင် နမူနာ ပစ္စည်း ၈ မျိုးကို ကြိုတင်ထည့်သွင်းထားပါတယ်
2. Menu System

- displayMenu() function မှ menu ကို ပြသပေးပါတယ်
- User က 1-4 အထိ နံပါတ်တစ်ခုကို ရွေးချယ်ရပါမယ်

3. လုပ်ဆောင်ချက်များ
   - Option 1: ပစ္စည်းအားလုံးကို မူလအတိုင်း ပြသမည်
   - Option 2: Category အလိုက်စီပြီး ပြသမည် (တူညီသော Category ရှိပါက Name အလိုက်စီပါမယ်)
   - Option 3: Price အလိုက် ဈေးနည်းရာမှ အများဆီ စီပြမည်
   - Option 4: Program မှ ထွက်ခွာမည်
4. ရလဒ်များ ပြသခြင်း
   - displayStockTable() function က ဇယားပုံစံဖြင့် စနစ်တကျ ပြသပေးပါမယ်
   - ID, Name, Category, Price နှင့် Quantity တို့ကို ဖော်ပြပေးပါမယ်
   - စုစုပေါင်း ပစ္စည်းအရေအတွက်ကို အောက်ဆုံးတွင် ဖော်ပြပေးပါမယ်

✓ ဥပမာ Run ကြည့်ခြင်း

```
==================================
    Stock Item Management System
==================================
1. Stock List အားလုံးကိုကြည့်မည်
2. Category အလိုက်စီပြမည်
3. Price အလိုက်စီပြမည်
4. Program မှ ထွက်ခွာမည်
==================================
လုပ်ဆောင်ချက် ရွေးချယ်ပါ (1-4) : 2
```

```
Stock Item List:
------------------------------------------------------------------------
ID        Name                Category       Price    Qty
------------------------------------------------------------------------
2001     T-Shirt             Clothing       $19.99    50
2002     Jeans               Clothing       $49.99    25
1001     Laptop              Electronics    $1200.99  15
1002     Smartphone          Electronics    $899.50   30
3001     Coffee              Groceries      $8.99     100
3002     Tea                 Groceries      $5.99     80
4001     Hammer              Tools          $12.50    40
4002     Screwdriver         Tools          $7.99     60
------------------------------------------------------------------------
စုစုပေါင်း 8 မျိုးတွေ့ရှိပါသည်။

Enter နှိပ်ပါ ဆက်လက်ဆောင်ရွက်ရန်...
```

✓ နောက်ထပ်ဖြည့်စွက်နိုင်သည့် Features များ

1. File I/O
    - Stock data များကို file မှ ဖတ်ခြင်း/ရေးခြင်း
    - CSV သို့မဟုတ် text file format ဖြင့် သိမ်းဆည်းခြင်း
2. ပစ္စည်းအသစ်ထည့်ခြင်း/ဖျက်ခြင်း
    - User က ပစ္စည်းအသစ်များ ထည့်သွင်းနိုင်ခြင်း
    - ရှိပြီးသား ပစ္စည်းများကို ဖျက်နိုင်ခြင်း
3. အဆင့်မြှင့် Sorting
    - ဈေးမြင့်ရာမှ နည်းရာသို့ စီနိုင်ခြင်း
    - Quantity အလိုက် စီနိုင်ခြင်း
4. Search Functionality

- ID, Name သို့မဟုတ် Category အလိုက် ရှာဖွေနိုင်ခြင်း

မှတ်ချက်: ဒီ program ကို C++11 သို့မဟုတ် နောက်ပိုင်း version များဖြင့် compile လုပ်ရန် လိုအပ်ပါမယ်။ Windows တွင် system("cls") ကို အသုံးပြုပြီး Linux/macOS တွင် system("clear") ကို အသုံးပြုပါ။

---

friend Keyword

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ friend function and friend classes

C++ programming မှာ friend keyword က class တစ်ခုရဲ့ private နှင့် protected members တွေကို အခြား functions သို့မဟုတ် classes များက ဝင်ရောက်အသုံးပြုနိုင်စေဖို့ ခွင့်ပြုပေးတဲ့ mechanism တစ်ခုဖြစ်ပါတယ်။

✓ friend ၏ အခြေခံအလုပ်လုပ်ပုံ

friend declaration ကို class ထဲမှာ သတ်မှတ်ပေးရပါမယ်။ ဒီလိုသတ်မှတ်လိုက်ခြင်းအားဖြင့် friend ဖြစ်သွားတဲ့ function သို့မဟုတ် class က ဒီ class ရဲ့ private members တွေကို ဝင်ရောက်အသုံးပြုနိုင်ပါတယ်။

```
class MyClass {
private:
    int secretData;

public:
    MyClass() : secretData(0) {}

    // friend function declaration
    friend void showSecret(MyClass obj);
};

// friend function definition
void showSecret(MyClass obj) {
    cout << "Secret data is: " << obj.secretData << endl;
    // private member ကို ဝင်ရောက်အသုံးပြုနိုင်ပါတယ်
}
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ friend အမျိုးအစားများ

1. friend Functions

```
class Box {
private:
    double width;

public:
    Box(double w) : width(w) {}

    // friend function declaration
    friend void printWidth(Box box);
};

// friend function definition
void printWidth(Box box) {
    // private member ကို ဝင်ရောက်အသုံးပြုနိုင်ပါတယ်
    cout << "Width of box: " << box.width << endl;
}

int main() {
    Box box(10.0);
    printWidth(box); // friend function ကို ခေါ်ယူအသုံးပြုခြင်း
    return 0;
}
```

2. friend Classes

```
class ClassA {
private:
    int secretValue;

public:
    ClassA() : secretValue(42) {}

    // ClassB ကို friend class အဖြစ် သတ်မှတ်ခြင်း
```

```
    friend class ClassB;
};

class ClassB {
public:
    void showSecret(ClassA a) {
        // ClassA ရဲ့private member ကို ဝင်ရောက်အသုံးပြုနိုင်ပါတယ်
        cout << "ClassA's secret: " << a.secretValue << endl;
    }
};

int main() {
    ClassA a;
    ClassB b;
    b.showSecret(a);
    return 0;
}
```

3. friend Member Functions

```
class ClassX; // forward declaration

class ClassY {
public:
    void display(ClassX x);
};

class ClassX {
private:
    int data;

public:
    ClassX() : data(100) {}

    // ClassY ရဲ့display function ကို friend အဖြစ် သတ်မှတ်ခြင်း
    friend void ClassY::display(ClassX x);
};
```

```
// ClassY::display definition (ClassX ကို အပြည့်အစုံသိပြီးမှသာ define လုပ်ရမယ်)
void ClassY::display(ClassX x) {
    cout << "ClassX data: " << x.data << endl;
}

int main() {
    ClassX x;
    ClassY y;
    y.display(x);
    return 0;
}
```

✓ friend ၏ အရေးကြီးသော အချက်များ

1. ဆက်ဆံရေး သဘောသဘာဝ
    - Friend relationship က နှစ်ဘက်ညီညီ (two-way) မဟုတ်ပါဘူး
    - Class A က Class B ကို friend လုပ်ပေးရင် Class B က Class A ရဲ့ private members တွေကို ဝင်ကြည့်နိုင်ပေမယ့် Class A က Class B ရဲ့ private members တွေကို ဝင်ကြည့်ခွင့် မရှိပါဘူး
2. Inheritance နှင့် ဆက်စပ်မှု
    - Friend relationship က inheritance နဲ့ မဆက်စပ်ပါဘူး
    - Base class ရဲ့ friend က derived class ရဲ့ private members တွေကို ဝင်ကြည့်ခွင့် မရှိပါဘူး
3. အသုံးပြုသင့်သည့် အခြေအနေများ
    - Operator overloading လုပ်တဲ့အခါမျိုး
    - ဆက်စပ်နေတဲ့ classes တွေကြားမှာ ချိတ်ဆက်အလုပ်လုပ်တဲ့အခါ
    - Unit testing အတွက် private members တွေကို test လုပ်ဖို့

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ friend ၏ ကောင်းကျိုးများ

1. Encapsulation ကို ချိုးဖောက်ရန် လိုအပ်သော အခြေအနေများတွင် အသုံးဝင်ပါတယ်
2. Operator overloading လုပ်တဲ့အခါ အထူးသင့်လျော်ပါတယ်
3. အချို့သော design patterns တွေမှာ အသုံးဝင်ပါတယ်

✓ Friend ၏ ဆိုးကျိုးများ

1. Encapsulation နည်းလမ်းကို အားနည်းစေပါတယ်
2. Code ကို maintain လုပ်ရခက်စေနိုင်ပါတယ်
3. Overuse လုပ်မိရင် design ကို ပျက်စီးစေနိုင်ပါတယ်

✓ ဥပမာ - Operator Overloading တွင် friend အသုံးပြုခြင်း

```
class Distance {
private:
    int meters;

public:
    Distance(int m = 0) : meters(m) {}

    // friend function for operator overloading
    friend Distance operator+(const Distance& d1, const Distance& d2);

    void display() {
        cout << "Meters: " << meters << endl;
    }
};

// + operator overload using friend function
Distance operator+(const Distance& d1, const Distance& d2) {
    return Distance(d1.meters + d2.meters);
```

```
}

int main() {
    Distance d1(10);
    Distance d2(20);
    Distance d3 = d1 + d2; // friend function ကို အသုံးပြုခြင်း

    d3.display(); // 30 ကို ပြသပေးပါမယ်
    return 0;
}
```

မှတ်ချက်: friend mechanism ကို သေချာစဉ်းစားပြီးမှသာ အသုံးပြုသင့်ပါတယ်။ မလိုအပ်ဘဲ အလွန်အကျွံသုံးစွဲမိရင် object-oriented design principles တွေကို ထိခိုက်စေနိုင်ပါတယ်။

✓ လေ့ကျင့်ရန်

ဒီ application မှာ Dijkstra's algorithm ကို အသုံးပြုပြီး graph ထဲက node တွေကြားမှာ shortest path ကို တွက်ချက်ပေးပါမယ်။ friend class ကို အသုံးပြုပြီး Graph class ရဲ့ private members တွေကို ဝင်ရောက်အသုံးပြုထားပါတယ်။

✓ Program Code

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <unordered_map>
using namespace std;

// Forward declaration
class DijkstraAlgorithm;
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
class Graph {
private:
    int V; // Number of vertices
    unordered_map<int, vector<pair<int, int>>> adjList; // Adjacency list

    // DijkstraAlgorithm class ကို friend အဖြစ် သတ်မှတ်ခြင်း
    friend class DijkstraAlgorithm;

public:
    Graph(int vertices) : V(vertices) {}

    void addEdge(int u, int v, int weight) {
        adjList[u].push_back({v, weight});
        adjList[v].push_back({u, weight}); // Undirected graph အတွက်
    }
};

class DijkstraAlgorithm {
public:
    static void findShortestPath(Graph& graph, int start) {
        // Priority queue for min-heap (distance, vertex)
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<i
nt, int>>> pq;

        // Distance vector - initialized to infinity
        unordered_map<int, int> dist;
        for(auto& pair : graph.adjList) {
            dist[pair.first] = INT_MAX;
        }

        // Parent map to store path
        unordered_map<int, int> parent;

        // Start with the source vertex
        dist[start] = 0;
        pq.push({0, start});
        parent[start] = -1;

        while(!pq.empty()) {
            int u = pq.top().second;
            pq.pop();
```

```
            // Visit all neighbors of u
            for(auto& neighbor : graph.adjList[u]) {
                int v = neighbor.first;
                int weight = neighbor.second;

                // Relaxation step
                if(dist[v] > dist[u] + weight) {
                    dist[v] = dist[u] + weight;
                    parent[v] = u;
                    pq.push({dist[v], v});
                }
            }
        }

        // Print shortest distances and paths
        printResults(dist, parent, start);
    }

private:
    static void printResults(unordered_map<int, int>& dist, unordered_map<int
, int>& parent, int start) {
        cout << "\nShortest paths from vertex " << start << ":\n";
        cout << "Vertex\tDistance\tPath\n";

        for(auto& d : dist) {
            int v = d.first;
            cout << v << "\t" << d.second << "\t\t";

            // Print path
            printPath(parent, v);
            cout << endl;
        }
    }

    static void printPath(unordered_map<int, int>& parent, int v) {
        if(v == -1) return;
        printPath(parent, parent[v]);
        cout << v << " ";
    }
};

int main() {
```

```
    // Create a graph
    Graph g(6);

    // Add edges
    g.addEdge(0, 1, 4);
    g.addEdge(0, 2, 1);
    g.addEdge(1, 2, 2);
    g.addEdge(1, 3, 5);
    g.addEdge(2, 3, 8);
    g.addEdge(2, 4, 10);
    g.addEdge(3, 4, 2);
    g.addEdge(3, 5, 6);
    g.addEdge(4, 5, 3);

    cout << "Dijkstra's Shortest Path Algorithm Implementation\n";
    cout << "Graph contains vertices 0 to 5 with following edges:\n";
    cout << "0-1(4), 0-2(1), 1-2(2), 1-3(5), 2-3(8), 2-4(10), 3-4(2), 3-5(6),
4-5(3)\n";

    int startVertex;
    cout << "\nEnter starting vertex (0-5): ";
    cin >> startVertex;

    // Find shortest paths using Dijkstra's algorithm
    DijkstraAlgorithm::findShortestPath(g, startVertex);

    return 0;
}
```

✓ Program အလုပ်လုပ်ပုံ

1. Graph ဖန်တီးခြင်း
    - Graph class မှာ vertices အရေအတွက်နဲ့ adjacency list ကို သိမ်းဆည်းထားပါတယ်
    - addEdge method ကို အသုံးပြုပြီး edges တွေကို ထည့်သွင်းနိုင်ပါတယ်
2. Dijkstra's Algorithm အကောင်အထည်ဖော်ခြင်း
    - DijkstraAlgorithm class က Graph class ရဲ့ friend ဖြစ်တဲ့အတွက် Graph ရဲ့ private members တွေကို ဝင်ရောက်အသုံးပြုနိုင်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

- Priority queue (min-heap) ကို အသုံးပြုပြီး shortest path တွေကို တွက်ချက်ပါတယ်
- Relaxation step မှာ current distance နဲ့ new distance ကို နှိုင်းယှဉ်ပြီး update လုပ်ပါတယ်

3. ရလဒ်များကို ပြသခြင်း
   - စတင်ခဲ့တဲ့ vertex ကနေ အခြား vertices တွေဆီကို shortest distance တွေနဲ့ path တွေကို ပြသပေးပါတယ်

✓ ဥပမာ Run ကြည့်ခြင်း

```
Dijkstra's Shortest Path Algorithm Implementation
Graph contains vertices 0 to 5 with following edges:
0-1(4), 0-2(1), 1-2(2), 1-3(5), 2-3(8), 2-4(10), 3-4(2), 3-5(6), 4-5(3)

Enter starting vertex (0-5): 0

Shortest paths from vertex 0:
Vertex  Distance        Path
0       0               0
1       3               0 2 1
2       1               0 2
3       8               0 2 1 3
4       10              0 2 1 3 4
5       13              0 2 1 3 4 5
```

✓ Friend Class ၏ အရေးပါမှု

1. Encapsulation ကို မထိခိုက်စေဘဲ လိုအပ်သော access ရရှိခြင်း
   - DijkstraAlgorithm class က Graph class ရဲ့ private adjList ကို ဝင်ရောက်ဖတ်နိုင်ပါတယ်
   - ဒါပေမယ့် တခြား class တွေကတော့ ဝင်ရောက်ဖတ်လို့ မရပါဘူး
2. Clean Interface ရရှိခြင်း

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

- Graph implementation details တွေကို ဖော်ပြစရာမလိုဘဲ algorithm ကို သပ်သပ်ရပ်ရပ် ရေးသားနိုင်ပါတယ်

3. Performance Optimization
   - Private members တွေကို directly access လုပ်နိုင်တဲ့အတွက် performance ပိုကောင်းပါတယ်

Dijkstra's Algorithm ၏ အဆင့်များ

1. Initialization
   - စတင်မယ့် vertex ရဲ့ distance ကို 0 ထားပြီး ကျန်တဲ့ vertices တွေကို infinity ထားပါတယ်
2. Priority Queue အသုံးပြုခြင်း
   - လက်ရှိအခြေအနေအရ shortest path ရှိတဲ့ vertex ကို ဦးစားပေး ရွေးချယ်ပါတယ်
3. Relaxation
   - Neighbor vertices တွေရဲ့ distance တွေကို update လုပ်ပါတယ်
   - လက်ရှိ distance ထက် ပိုကောင်းတဲ့ path တွေ့ရင် update လုပ်ပါတယ်
4. Result Output
   - ရလဒ်တွေကို ဖော်ပြပါတယ်

မှတ်ချက်: ဒီ implementation မှာ undirected graph ကို အသုံးပြုထားပြီး vertices တွေကို numbers တွေနဲ့ ကိုယ်စားပြုထားပါတယ်။ Real-world application တွေမှာ city names သို့မဟုတ် location IDs တွေကို အသုံးပြုနိုင်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Templates

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ Templates

C++ Templates ဆိုတာ generic programming အတွက် အခြေခံအုတ်မြစ်ဖြစ်ပြီး၊ မတူညီတဲ့ data types တွေအတွက် reusable code တွေရေးဖို့ အသုံးပြုပါတယ်။

✓ Templates အမျိုးအစားများ

1. Function Templates (လုပ်ဆောင်ချက်ပုံစံများ)

```
template <typename T>
T findMax(T x, T y) {
    return (x > y) ? x : y;
}

// အသုံးပြုပုံ
int maxInt = findMax(3, 7);           // int type
double maxDouble = findMax(3.14, 2.99); // double type
```

2. Class Templates (အတန်းပုံစံများ)

```
template <class T>
class Stack {
private:
    vector<T> elements;
public:
    void push(T const& elem) {
        elements.push_back(elem);
    }
    void pop() {
        elements.pop_back();
    }
    T top() const {
        return elements.back();
    }
};
```

```
// အသုံးပြုပုံ
Stack<int> intStack;    // integer stack
Stack<string> strStack; // string stack
```

✓ Template Parameters အမျိုးအစားများ

1. Type Parameters:

```
template <typename T, typename U>
```

2. Non-type Parameters:

```
template <int size>
class Array {
    int arr[size];
};
```

3. Template template parameters:

```
template <template <typename> class Container>
class Grid {
    Container<int> cells;
};
```

✓ Template Specialization (အထူးသတ်မှတ်ခြင်း)

1. Full Specialization:

```
template <>
class Stack<bool> {
    // bool type အတွက် အထူးအလုပ်လုပ်မယ်
};
```

2. Partial Specialization:

```
template <typename T>
class Stack<T*> {
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
    // pointer types တွေအတွက် အထူးအလုပ်လုပ်မယ်
};
```

✓ Template Metaprogramming (အဆင့်မြင့် အသုံးပြုနည်း)

Compile-time မှာတင် တွက်ချက်မှုတွေလုပ်နိုင်တဲ့ technique:

```
template <int n>
struct Factorial {
    enum { value = n * Factorial<n-1>::value };
};

template <>
struct Factorial<0> {
    enum { value = 1 };
};

// အသုံးပြုပုံ
int fact5 = Factorial<5>::value;  // 120
```

✓ STL နဲ့ Templates

Standard Template Library (STL) တစ်ခုလုံးနီးပါးက templates တွေကိုအခြေခံပြီးတည်ဆောက်ထားတာ ဖြစ်ပါတယ်။

- vector<T>, list<T>, map<K,V> - Container templates
- sort(), find() - Algorithm templates
- iterator - Iterator templates

✓ ကောင်းကျိုးများ

1. Code Reusability - တစ်ခါရေးပြီး အကြိမ်ကြိမ်သုံးနိုင်
2. Type Safety - Type checking ကို compile-time မှာတင်လုပ်နိုင်
3. Performance - Run-time overhead မရှိ
4. Flexibility - မတူညီတဲ့ data types တွေအတွက် အလုပ်လုပ်နိုင်

✓ Short Notes:

1. Template definition တွေကို header files ထဲမှာပဲထားသင့်တယ်
2. Complex templates တွေက compile error တွေကိုနားလည်ရခက်နိုင်တယ်
3. Template instantiation ကြောင့် code bloat ဖြစ်နိုင်တယ်

Templates တွေကို ကျွမ်းကျွမ်းကျင်ကျင်အသုံးပြုနိုင်ရင် C++ programming မှာ အင်မတန်ထိရောက်တဲ့ generic code တွေရေးနိုင်မှာဖြစ်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန်

❒ Tower of Hanoi Application

Tower of Hanoi ပဟေဠိအတွက် generic template version ကို အောက်တွင်ဖော်ပြထားပါတယ်။ ဒီ program မှာ template ကိုသုံးပြီး disk အမျိုးအစားအမျိုးမျိုးနဲ့ အလုပ်လုပ်နိုင်အောင် ရေးထားပါတယ်။

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

// Tower အတွက် template class
template <typename T>
class Tower {
private:
    vector<T> disks;
    string name;

public:
    Tower(string n) : name(n) {}

    void push(T disk) {
        disks.push_back(disk);
    }

    T pop() {
        T top = disks.back();
        disks.pop_back();
        return top;
    }

    string getName() const {
        return name;
    }
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

```
    void print() const {
        cout << name << " Tower: ";
        for (const auto& disk : disks) {
            cout << disk << " ";
        }
        cout << endl;
    }
};

// Hanoi puzzle ဖြေရှင်းမည့် template function
template <typename T>
void moveDisks(int n, Tower<T>& source, Tower<T>& target, Tower<T>& auxiliary
) {
    if (n > 0) {
        // n-1 disks ကို source ကနေ auxiliary ကိုရွှေ့မယ်
        moveDisks(n - 1, source, auxiliary, target);

        // ကျန်တဲ့ disk ကို source ကနေ target ကိုရွှေ့မယ်
        T disk = source.pop();
        target.push(disk);
        cout << "Disk " << disk << " ကို " << source.getName()
             << " မှ " << target.getName() << " သို့ ရွှေ့ပါတယ်" << endl;

        // n-1 disks ကို auxiliary ကနေ target ကိုရွှေ့မယ်
        moveDisks(n - 1, auxiliary, target, source);
    }
}

int main() {
    // int disks နဲ့အလုပ်လုပ်တဲ့ဥပမာ
    {
        Tower<int> towerA("A");
        Tower<int> towerB("B");
        Tower<int> towerC("C");

        int numDisks = 3;
        for (int i = numDisks; i >= 1; --i) {
            towerA.push(i);
        }
```

```
        cout << "အစဦး အခြေအနေ:" << endl;
        towerA.print();
        towerB.print();
        towerC.print();
        cout << endl;

        cout << "ရွှေ့ပြောင်းမှု အဆင့်များ:" << endl;
        moveDisks(numDisks, towerA, towerC, towerB);

        cout << "\nနောက်ဆုံး အခြေအနေ:" << endl;
        towerA.print();
        towerB.print();
        towerC.print();
    }

    cout << "\n--------------------------------\n" << endl;

    // char disks နဲ့အလုပ်လုပ်တဲ့ဥပမာ
    {
        Tower<char> towerX("X");
        Tower<char> towerY("Y");
        Tower<char> towerZ("Z");

        towerX.push('L'); // အကြီးဆုံး disk
        towerX.push('M');
        towerX.push('S'); // အသေးဆုံး disk

        cout << "အစဦး အခြေအနေ:" << endl;
        towerX.print();
        towerY.print();
        towerZ.print();
        cout << endl;

        cout << "ရွှေ့ပြောင်းမှု အဆင့်များ:" << endl;
        moveDisks(3, towerX, towerZ, towerY);
    }

    return 0;
}
```

✓ Program အလုပ်လုပ်ပုံ

1. Template Class Tower<T>:
    - Disk တွေကို သိမ်းဖို့ vector<T> သုံးထားတယ်
    - Disk တွေကို push/pop လုပ်ဖို့ method တွေပါဝင်တယ်
    - Tower အခြေအနေကို print ထုတ်ဖို့ method ပါတယ်
2. Template Function moveDisks<T>:
    - Recursive algorithm နဲ့ Tower of Hanoi ပဟေဋ္ဌိကိုဖြေရှင်းတယ်
    - n disks ကို source tower ကနေ target tower ကိုရွှေ့တယ်
    - auxiliary tower ကိုအကူအညီအနေနဲ့သုံးတယ်
3. Main Function:
    - နှစ်မျိုးနဲ့ပြထားတယ် - integer disks နဲ့ char disks
    - အစဦးအခြေအနေ၊ ရွှေ့ပြောင်းမှုအဆင့်တွေ၊ နောက်ဆုံးအခြေအနေကိုပြသထားတယ်

✓ ဥပမာ Output

```
အစဦး အခြေအနေ:
A Tower: 3 2 1
B Tower:
C Tower:

ရွှေ့ပြောင်းမှု အဆင့်များ:
Disk 1 ကို A မှ C သို့ ရွှေ့ပါတယ်
Disk 2 ကို A မှ B သို့ ရွှေ့ပါတယ်
Disk 1 ကို C မှ B သို့ ရွှေ့ပါတယ်
Disk 3 ကို A မှ C သို့ ရွှေ့ပါတယ်
Disk 1 ကို B မှ A သို့ ရွှေ့ပါတယ်
Disk 2 ကို B မှ C သို့ ရွှေ့ပါတယ်
Disk 1 ကို A မှ C သို့ ရွှေ့ပါတယ်

နောက်ဆုံး အခြေအနေ:
```

```
A Tower:
B Tower:
C Tower: 3 2 1

--------------------------------

အစဦး အခြေအနေ:
X Tower: S M L
Y Tower:
Z Tower:

ရွေ့ပြောင်းမှု အဆင့်များ:
Disk S ကို X မှ Z သို့ ရွှေ့ပါတယ်
Disk M ကို X မှ Y သို့ ရွှေ့ပါတယ်
Disk S ကို Z မှ Y သို့ ရွှေ့ပါတယ်
Disk L ကို X မှ Z သို့ ရွှေ့ပါတယ်
Disk S ကို Y မှ X သို့ ရွှေ့ပါတယ်
Disk M ကို Y မှ Z သို့ ရွှေ့ပါတယ်
Disk S ကို X မှ Z သို့ ရွှေ့ပါတယ်
```

✓ Template သုံးခြင်း၏ အားသာချက်များ

1. Code Reusability - တစ်ခါရေးပြီး မတူညီတဲ့ data types တွေနဲ့သုံးနိုင်တယ်
2. Type Safety - Compile-time မှာ type checking လုပ်ပေးတယ်
3. Clean Code - အထူးသတ်မှတ်ထားတဲ့ type တွေအတွက် အပိုလိုအပ်ချက်မရှိ

ဒီ program မှာ template ကိုသုံးပြီး Tower of Hanoi algorithm ကို generic ဖြစ်အောင်ရေးထားတာကို မြင်တွေ့နိုင်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

FILE

❒ File

C++ မှာ file တွေကို read/write လုပ်ဖို့ <fstream> library ကို အသုံးပြုပါတယ်။ File operations တွေအတွက် အဓိက class ၃ ခုရှိပါတယ်:

1. ofstream - File ကို write လုပ်ဖို့ (Output File Stream)
2. ifstream - File ကို read လုပ်ဖို့ (Input File Stream)
3. fstream - Read/Write နှစ်မျိုးလုံးလုပ်ဖို့ (File Stream)

✓ File ရေးသားခြင်း (Writing to File)

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    // ofstream object ဖန်တီးပါတယ်
    ofstream outFile("data.txt");

    // File ဖွင့်နိုင်ခြေ ရှိ/မရှိ စစ်ပါတယ်
    if(!outFile) {
        cout << "File ဖွင့်ရာတွင် အမှားတစ်ခုဖြစ်နေပါတယ်" << endl;
        return 1;
    }

    // File ထဲကို data တွေ ရေးသားပါတယ်
    outFile << "မင်္ဂလာပါ C++ File Handling" << endl;
    outFile << "ဤသည်မှာ စာသားဖိုင်တစ်ခုဖြစ်ပါတယ်" << endl;
    outFile << 12345 << endl;
    outFile << 67.89 << endl;

    // File ကို ပိတ်ပါတယ်
    outFile.close();
```

```
    cout << "File ကို အချက်အလက် ရေးသားပြီးပါပြီ" << endl;
    return 0;
}
```

✓ File ဖတ်ခြင်း (Reading from File)

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    // ifstream object ဖန်တီးပါတယ်
    ifstream inFile("data.txt");

    // File ဖွင့်နိုင်ခြေ ရှိ/မရှိ စစ်ပါတယ်
    if(!inFile) {
        cout << "File ဖွင့်ရာတွင် အမှားတစ်ခုဖြစ်နေပါတယ်" << endl;
        return 1;
    }

    string line;
    int number;
    double value;

    // File ထဲက data တွေကို line by line ဖတ်ပါတယ်
    while(getline(inFile, line)) {
        cout << line << endl;
    }

    // File position ကို ပြန်စပါတယ်
    inFile.clear();
    inFile.seekg(0);

    // Specific data types တွေကို ဖတ်ပါတယ်
    inFile >> number >> value;
    cout << "နံပါတ်: " << number << endl;
    cout << "တန်ဖိုး: " << value << endl;
```

```
    // File ကို ပိတ်ပါတယ်
    inFile.close();

    return 0;
}
```

✓ File Modes (ဖိုင်ဖွင့်နည်းများ)

File တွေကို အမျိုးမျိုးသော mode တွေနဲ့ ဖွင့်နိုင်ပါတယ်:

```
// ios::out - Write mode (default for ofstream)
// ios::in - Read mode (default for ifstream)
// ios::app - Append mode (ဖိုင်အဆုံးမှာ ထပ်ထည့်ရန်)
// ios::ate - Open and seek to end of file
// ios::trunc - File ကို overwrite လုပ်မယ် (default)
// ios::binary - Binary mode နဲ့ ဖွင့်မယ်

ofstream outFile("data.txt", ios::app);  // Append mode
fstream ioFile("data.txt", ios::in | ios::out);  // Read and Write mode
```

✓ Binary File Operations

Binary file တွေကို read/write လုပ်တဲ့အခါမှာ read() နဲ့ write() functions တွေကို အသုံးပြုပါတယ်။

```
#include <iostream>
#include <fstream>
using namespace std;

struct Student {
    int id;
    char name[50];
    double gpa;
};
```

```
int main() {
    // Binary mode နဲ့ file ရေးသားခြင်း
    ofstream outFile("students.dat", ios::binary);

    Student s1 = {101, "မောင်မောင်", 3.8};
    Student s2 = {102, "စုစုလှိုင်", 4.0};

    outFile.write(reinterpret_cast<char*>(&s1), sizeof(Student));
    outFile.write(reinterpret_cast<char*>(&s2), sizeof(Student));
    outFile.close();

    // Binary mode နဲ့ file ဖတ်ခြင်း
    ifstream inFile("students.dat", ios::binary);
    Student temp;

    while(inFile.read(reinterpret_cast<char*>(&temp), sizeof(Student))) {
        cout << "ID: " << temp.id << endl;
        cout << "Name: " << temp.name << endl;
        cout << "GPA: " << temp.gpa << endl << endl;
    }

    inFile.close();
    return 0;
}
```

✓ File Handling Tips

1. File operations တွေလုပ်တဲ့အခါမှာ အမြဲတမ်း file ကို ဖွင့်နိုင်ခြေ ရှိ/မရှိ စစ်ပါတယ်
2. File တွေကို အသုံးပြုပြီးတဲ့အခါမှာ အမြဲတမ်း close လုပ်ပါတယ်
3. Binary files တွေကို ဖတ်တဲ့အခါမှာ data structure တွေရဲ့ size ကို သေချာစစ်ပါတယ်
4. Exception handling တွေကို အသုံးပြုပြီး error တွေကို handle လုပ်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန်

❒ Yangon Hotel Management System

```
yangon_hotel/
├─── include/
│    ├─── Room.h
│    ├─── Guest.h
│    ├─── Booking.h
│    └─── HotelSystem.h
├─── src/
│    ├─── Room.cpp
│    ├─── Guest.cpp
│    ├─── Booking.cpp
│    ├─── HotelSystem.cpp
│    └─── main.cpp
├─── data/
│    ├─── rooms.dat
│    ├─── guests.dat
│    └─── bookings.dat
└─── Makefile
```

✓ Header Files

1. Room.h

```
#ifndef ROOM_H
#define ROOM_H

#include <string>
#include <iostream>
#include <iomanip>
```

```
enum RoomType { SINGLE, DOUBLE, SUITE };
```

```
class Room {
private:
    int roomNumber;
    RoomType type;
    bool isBooked;
    double pricePerNight;

public:
    Room(int num = 0, RoomType t = SINGLE, double price = 0.0);

    // Getter methods
    int getRoomNumber() const;
    RoomType getType() const;
    bool getIsBooked() const;
    double getPrice() const;

    // Setter methods
    void bookRoom();
    void releaseRoom();

    std::string getTypeString() const;
    void display() const;
};

#endif
```

2. Guest.h

```
#ifndef GUEST_H
#define GUEST_H

#include <string>
#include <iostream>

class Guest {
private:
    std::string name;
    std::string phone;
    std::string address;
    std::string nrc;
```

```
public:
    Guest(const std::string& n = "", const std::string& p = "",
          const std::string& addr = "", const std::string& id = "");

    // Getter methods
    std::string getName() const;
    std::string getPhone() const;
    std::string getAddress() const;
    std::string getNRC() const;

    void display() const;
};

#endif
```

3. Booking.h

```
#ifndef BOOKING_H
#define BOOKING_H

#include <ctime>
#include <string>
#include <iostream>

class Booking {
private:
    int roomNumber;
    int guestIndex;
    time_t checkInDate;
    time_t checkOutDate;
    int nights;
    double totalAmount;
    bool isPaid;

public:
    Booking(int room = 0, int guest = -1, int nights = 1);

    // Getter methods
```

```
    int getRoomNumber() const;
    int getGuestIndex() const;
    time_t getCheckInDate() const;
    time_t getCheckOutDate() const;
    int getNights() const;
    double getTotalAmount() const;
    bool getIsPaid() const;

    // Setter methods
    void setCheckOutDate(time_t date);
    void setTotalAmount(double amount);
    void markAsPaid();

    std::string getDateString(time_t date) const;
    void display() const;
};

#endif
```

4. HotelSystem.h

```
#ifndef HOTEL_SYSTEM_H
#define HOTEL_SYSTEM_H

#include <vector>
#include <string>
#include <fstream>
#include "Room.h"
#include "Guest.h"
#include "Booking.h"

class HotelManagementSystem {
private:
    std::vector<Room> rooms;
    std::vector<Guest> guests;
    std::vector<Booking> bookings;

    const std::string ROOMS_FILE = "data/rooms.dat";
    const std::string GUESTS_FILE = "data/guests.dat";
    const std::string BOOKINGS_FILE = "data/bookings.dat";
```

```
    void loadData();
    void saveData();

public:
    HotelManagementSystem();
    ~HotelManagementSystem();

    void initializeRooms();
    void displayAllRooms() const;
    int registerGuest();
    void bookRoom();
    void processPayment();
    void displayBookings() const;
};

#endif
```

✓ Source Files

1. Room.cpp

```
#include "Room.h"

Room::Room(int num, RoomType t, double price)
    : roomNumber(num), type(t), isBooked(false), pricePerNight(price) {}

int Room::getRoomNumber() const { return roomNumber; }
RoomType Room::getType() const { return type; }
bool Room::getIsBooked() const { return isBooked; }
double Room::getPrice() const { return pricePerNight; }

void Room::bookRoom() { isBooked = true; }
void Room::releaseRoom() { isBooked = false; }

std::string Room::getTypeString() const {
    switch(type) {
        case SINGLE: return "Single";
        case DOUBLE: return "Double";
```

```
        case SUITE: return "Suite";
        default: return "Unknown";
    }
}

void Room::display() const {
    std::cout << "| " << std::setw(4) << roomNumber << " | "
              << std::setw(10) << getTypeString() << " | "
              << std::setw(10) << (isBooked ? "Booked" : "Available") << " |
"
              << std::setw(8) << pricePerNight << " |" << std::endl;
}
```

2. Guest.cpp

```
#include "Guest.h"

Guest::Guest(const std::string& n, const std::string& p,
             const std::string& addr, const std::string& id)
    : name(n), phone(p), address(addr), nrc(id) {}

std::string Guest::getName() const { return name; }
std::string Guest::getPhone() const { return phone; }
std::string Guest::getAddress() const { return address; }
std::string Guest::getNRC() const { return nrc; }

void Guest::display() const {
    std::cout << "Guest Name: " << name << "\n"
              << "Phone: " << phone << "\n"
              << "Address: " << address << "\n"
              << "NRC: " << nrc << std::endl;
}
```

3. Booking.cpp

```
#include "Booking.h"

Booking::Booking(int room, int guest, int nights)
    : roomNumber(room), guestIndex(guest), nights(nights),
```

```
      isPaid(false) {
    checkInDate = time(nullptr);
    checkOutDate = 0;
    totalAmount = 0.0;
}

int Booking::getRoomNumber() const { return roomNumber; }
int Booking::getGuestIndex() const { return guestIndex; }
time_t Booking::getCheckInDate() const { return checkInDate; }
time_t Booking::getCheckOutDate() const { return checkOutDate; }
int Booking::getNights() const { return nights; }
double Booking::getTotalAmount() const { return totalAmount; }
bool Booking::getIsPaid() const { return isPaid; }

void Booking::setCheckOutDate(time_t date) { checkOutDate = date; }
void Booking::setTotalAmount(double amount) { totalAmount = amount; }
void Booking::markAsPaid() { isPaid = true; }

std::string Booking::getDateString(time_t date) const {
    if (date == 0) return "Not checked out";
    char buffer[80];
    strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", localtime(&date));
    return std::string(buffer);
}

void Booking::display() const {
    std::cout << "Room Number: " << roomNumber << "\n"
              << "Check-in Date: " << getDateString(checkInDate) << "\n"
              << "Check-out Date: " << getDateString(checkOutDate) << "\n"
              << "Nights: " << nights << "\n"
              << "Total Amount: $" << std::fixed << std::setprecision(2) << t
otalAmount << "\n"
              << "Payment Status: " << (isPaid ? "Paid" : "Pending") << std::
endl;
}
```

4. HotelSystem.cpp

```
#include "HotelSystem.h"
#include <ctime>
#include <limits>
```

```
HotelManagementSystem::HotelManagementSystem() {
    loadData();
}

HotelManagementSystem::~HotelManagementSystem() {
    saveData();
}

void HotelManagementSystem::loadData() {
    std::ifstream roomFile(ROOMS_FILE, std::ios::binary);
    std::ifstream guestFile(GUESTS_FILE, std::ios::binary);
    std::ifstream bookingFile(BOOKINGS_FILE, std::ios::binary);

    Room tempRoom;
    while (roomFile.read(reinterpret_cast<char*>(&tempRoom), sizeof(Room))) {
        rooms.push_back(tempRoom);
    }

    Guest tempGuest;
    while (guestFile.read(reinterpret_cast<char*>(&tempGuest), sizeof(Guest))
) {
        guests.push_back(tempGuest);
    }

    Booking tempBooking;
    while (bookingFile.read(reinterpret_cast<char*>(&tempBooking), sizeof(Boo
king))) {
        bookings.push_back(tempBooking);
    }
}

void HotelManagementSystem::saveData() {
    std::ofstream roomFile(ROOMS_FILE, std::ios::binary);
    std::ofstream guestFile(GUESTS_FILE, std::ios::binary);
    std::ofstream bookingFile(BOOKINGS_FILE, std::ios::binary);

    for (const auto& room : rooms) {
        roomFile.write(reinterpret_cast<const char*>(&room), sizeof(Room));
    }

    for (const auto& guest : guests) {
```

```
        guestFile.write(reinterpret_cast<const char*>(&guest), sizeof(Guest))
;
    }

    for (const auto& booking : bookings) {
        bookingFile.write(reinterpret_cast<const char*>(&booking), sizeof(Boo
king));
    }
}

void HotelManagementSystem::initializeRooms() {
    rooms.clear();
    rooms.emplace_back(101, SINGLE, 50.0);
    rooms.emplace_back(102, SINGLE, 50.0);
    rooms.emplace_back(201, DOUBLE, 80.0);
    rooms.emplace_back(202, DOUBLE, 80.0);
    rooms.emplace_back(301, SUITE, 150.0);
    rooms.emplace_back(302, SUITE, 150.0);
    std::cout << "Rooms initialized successfully.\n";
}

void HotelManagementSystem::displayAllRooms() const {
    std::cout << "\n=== Yangon Hotel Rooms ===\n";
    std::cout << "-----------------------------------\n";
    std::cout << "| Room | Type      | Status     | Price   |\n";
    std::cout << "-----------------------------------\n";

    for (const auto& room : rooms) {
        room.display();
    }

    std::cout << "-----------------------------------\n";
}

int HotelManagementSystem::registerGuest() {
    std::string name, phone, address, nrc;
    std::cin.ignore();

    std::cout << "Enter guest name: ";
    std::getline(std::cin, name);
    std::cout << "Enter phone number: ";
    std::getline(std::cin, phone);
```

```
    std::cout << "Enter address: ";
    std::getline(std::cin, address);
    std::cout << "Enter NRC number: ";
    std::getline(std::cin, nrc);

    guests.emplace_back(name, phone, address, nrc);
    return guests.size() - 1;
}

void HotelManagementSystem::bookRoom() {
    displayAllRooms();
    int roomNum, nights;

    std::cout << "Enter room number to book: ";
    std::cin >> roomNum;
    std::cout << "Enter number of nights: ";
    std::cin >> nights;

    bool found = false;
    for (auto& room : rooms) {
        if (room.getRoomNumber() == roomNum) {
            found = true;
            if (room.getIsBooked()) {
                std::cout << "Room is already booked!\n";
            } else {
                std::cout << "Registering guest...\n";
                int guestIndex = registerGuest();
                room.bookRoom();
                bookings.emplace_back(roomNum, guestIndex, nights);
                std::cout << "Booking successful!\n";
            }
            break;
        }
    }

    if (!found) {
        std::cout << "Invalid room number!\n";
    }
}

void HotelManagementSystem::processPayment() {
    int roomNum;
```

```
    std::cout << "Enter room number for checkout: ";
    std::cin >> roomNum;

    bool found = false;
    for (auto& booking : bookings) {
        if (booking.getRoomNumber() == roomNum && !booking.getIsPaid()) {
            found = true;

            Room* roomPtr = nullptr;
            for (auto& room : rooms) {
                if (room.getRoomNumber() == roomNum) {
                    roomPtr = &room;
                    break;
                }
            }

            if (roomPtr) {
                double total = roomPtr->getPrice() * booking.getNights();
                booking.setTotalAmount(total);
                booking.setCheckOutDate(time(nullptr));

                if (booking.getGuestIndex() >= 0 && booking.getGuestIndex() <
guests.size()) {
                    guests[booking.getGuestIndex()].display();
                }

                booking.display();

                char confirm;
                std::cout << "Confirm payment (Y/N)? ";
                std::cin >> confirm;

                if (toupper(confirm) == 'Y') {
                    booking.markAsPaid();
                    roomPtr->releaseRoom();
                    std::cout << "Payment processed successfully. Room releas
ed.\n";
                } else {
                    std::cout << "Payment cancelled.\n";
                }
            }
            break;
```

```
        }
    }

    if (!found) {
        std::cout << "No unpaid booking found for this room or invalid room n
umber.\n";
    }
}

void HotelManagementSystem::displayBookings() const {
    std::cout << "\n=== Current Bookings ===\n";
    if (bookings.empty()) {
        std::cout << "No current bookings.\n";
    } else {
        for (const auto& booking : bookings) {
            std::cout << "-----------------------\n";
            std::cout << "Room: " << booking.getRoomNumber() << "\n";

            if (booking.getGuestIndex() >= 0 && booking.getGuestIndex() < gue
sts.size()) {
                guests[booking.getGuestIndex()].display();
            }

            booking.display();
        }
    }
    std::cout << "=======================\n";
}
```

5. main.cpp

```
#include "HotelSystem.h"
#include <iostream>

void displayMainMenu() {
    std::cout << "\n=== Yangon Hotel Management System ===" << std::endl;
    std::cout << "1. Initialize Rooms" << std::endl;
    std::cout << "2. Display All Rooms" << std::endl;
    std::cout << "3. Book a Room" << std::endl;
    std::cout << "4. Process Payment (Checkout)" << std::endl;
    std::cout << "5. View Current Bookings" << std::endl;
    std::cout << "6. Exit" << std::endl;
```

```
    std::cout << "Enter your choice: ";
}

int main() {
    HotelManagementSystem hotelSystem;
    int choice;

    do {
        displayMainMenu();
        std::cin >> choice;

        switch(choice) {
            case 1:
                hotelSystem.initializeRooms();
                break;
            case 2:
                hotelSystem.displayAllRooms();
                break;
            case 3:
                hotelSystem.bookRoom();
                break;
            case 4:
                hotelSystem.processPayment();
                break;
            case 5:
                hotelSystem.displayBookings();
                break;
            case 6:
                std::cout << "Exiting system. Data saved automatically." << s
td::endl;
                break;
            default:
                std::cout << "Invalid choice. Please try again." << std::endl
;
        }
    } while (choice != 6);

    return 0;
}
```

✓ Makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -Iinclude
LDFLAGS =
SRCDIR = src
OBJDIR = obj
BINDIR = bin

SOURCES = $(wildcard $(SRCDIR)/*.cpp)
OBJECTS = $(patsubst $(SRCDIR)/%.cpp,$(OBJDIR)/%.o,$(SOURCES))
EXECUTABLE = $(BINDIR)/hotel_system

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
        @mkdir -p $(BINDIR)
        $(CXX) $(LDFLAGS) $^ -o $@

$(OBJDIR)/%.o: $(SRCDIR)/%.cpp
        @mkdir -p $(OBJDIR)
        $(CXX) $(CXXFLAGS) -c $< -o $@

clean:
        rm -rf $(OBJDIR) $(BINDIR)

run: $(EXECUTABLE)
        ./$(EXECUTABLE)

.PHONY: all clean run
```

✓ Testing Results

Test Case 1: Initialize Rooms

```
=== Yangon Hotel Management System ===
1. Initialize Rooms
2. Display All Rooms
3. Book a Room
4. Process Payment (Checkout)
5. View Current Bookings
6. Exit
Enter your choice: 1
Rooms initialized successfully.
```

Test Case 2: Display Rooms

```
Enter your choice: 2

=== Yangon Hotel Rooms ===
-----------------------------------
| Room | Type      | Status     | Price   |
-----------------------------------
|  101 | Single    | Available  |      50 |
|  102 | Single    | Available  |      50 |
|  201 | Double    | Available  |      80 |
|  202 | Double    | Available  |      80 |
|  301 | Suite     | Available  |     150 |
|  302 | Suite     | Available  |     150 |
-----------------------------------
```

Test Case 3: Book a Room

```
Enter your choice: 3

=== Yangon Hotel Rooms ===
-----------------------------------
```

```
| Room | Type      | Status     | Price   |
-------------------------------------
|  101 | Single    | Available  |      50 |
|  102 | Single    | Available  |      50 |
|  201 | Double    | Available  |      80 |
|  202 | Double    | Available  |      80 |
|  301 | Suite     | Available  |     150 |
|  302 | Suite     | Available  |     150 |
-------------------------------------
Enter room number to book: 201
Enter number of nights: 3
Registering guest...
Enter guest name: Aung Aung
Enter phone number: 09123456789
Enter address: No.123, Yangon
Enter NRC number: 12/YGN(N)123456
Booking successful!
```

Test Case 4: Process Payment

```
Enter your choice: 4
Enter room number for checkout: 201
Guest Name: Aung Aung
Phone: 09123456789
Address: No.123, Yangon
NRC: 12/YGN(N)123456
Room Number: 201
Check-in Date: 2023-06-15 14:30:45
Check-out Date: Not checked out
Nights: 3
Total Amount: $240.00
Payment Status: Pending

Confirm payment (Y/N)? Y
Payment processed successfully. Room released.
```

Test Case 5: View Bookings

```
Enter your choice: 5

=== Current Bookings ===
-----------------------
Room: 201
Guest Name: Aung Aung
Phone: 09123456789
Address: No.123, Yangon
NRC: 12/YGN(N)123456
Room Number: 201
Check-in Date: 2023-06-15 14:30:45
Check-out Date: 2023-06-15 14:31:10
Nights: 3
Total Amount: $240.00
Payment Status: Paid
=======================
```

Namespace

❒ Namespace

Namespace သည် C++ မှာ naming conflicts များကို ရှောင်ရှားရန်၊ code များကို logical group များအဖြစ် စုစည်းထားရန် အသုံးပြုပါတယ်။ အောက်တွင် namespace အသုံးပြုထားသော နမူနာများကို အသေးစိတ် ရှင်းပြထားပါတယ်။

1. ရိုးရှင်းသော Namespace နမူနာ

```
#include <iostream>

// သချိာဆိုင်ရာ လုပ်ဆောင်ချက်များအတွက် namespace
namespace Math {
    const double PI = 3.14159;

    double add(double a, double b) {
        return a + b;
    }

    double circleArea(double radius) {
        return PI * radius * radius;
    }
}

int main() {
    std::cout << "5 + 3 = " << Math::add(5, 3) << std::endl;
    std::cout << "Radius 2.0 circle area: " << Math::circleArea(2.0) << std::
endl;
    std::cout << "PI value: " << Math::PI << std::endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

- Math namespace အတွင်း PI constant နှင့် သခ်ၤာလုပ်ဆောင်ချက်များ ထည့်ထားပါတယ်။
- :: operator ဖြင့် namespace အတွင်းရှိ အရာများကို ခေါ်ယူနိုင်ပါတယ်။

2. Game Development အတွက် Namespace ဥပမာ

```
#include <iostream>
#include <string>

namespace GameEngine {
    namespace Graphics {
        void render(const std::string& object) {
            std::cout << "Rendering " << object << std::endl;
        }
    }

    namespace Physics {
        void update() {
            std::cout << "Updating physics" << std::endl;
        }
    }

    namespace Audio {
        void playSound(const std::string& sound) {
            std::cout << "Playing: " << sound << std::endl;
        }
    }
}

int main() {
```

```
    GameEngine::Graphics::render("Player Character");
    GameEngine::Physics::update();
    GameEngine::Audio::playSound("Explosion");

    return 0;
}
```

✓ ရှင်းလင်းချက်

- Game engine ၏ အစိတ်အပိုင်းများကို namespace အဆင့်ဆင့်ဖြင့် စနစ်တကျခွဲခြားထားပါတယ်။
- Graphics, Physics, Audio ကဲ့သို့သော module များကို သီးခြားစီ စီမံနိုင်ပါတယ်။

3. စာသားပြုပြင်ခြင်း Tools အတွက် Namespace

```
#include <iostream>
#include <string>
#include <algorithm>

namespace TextUtils {
    std::string toUpper(const std::string& text) {
        std::string result = text;
        std::transform(result.begin(), result.end(), result.begin(), ::touppe
r);
        return result;
    }

    std::string toLower(const std::string& text) {
        std::string result = text;
        std::transform(result.begin(), result.end(), result.begin(), ::tolowe
r);
        return result;
    }
}

int main() {
    std::string message = "Hello World";
```

```
    std::cout << "Original: " << message << std::endl;
    std::cout << "Uppercase: " << TextUtils::toUpper(message) << std::endl;
    std::cout << "Lowercase: " << TextUtils::toLower(message) << std::endl;

    return 0;
}
```

✓ ရှင်းလင်းချက်

- စာသားပြုပြင်ခြင်းဆိုင်ရာ function များကို `TextUtils` namespace အတွင်း စုစည်းထားပါတယ်။
- အခြား program များတွင် ဤ function များကို အလွယ်တကူ ပြန်လည်အသုံးပြုနိုင်ပါတယ်။

4. Anonymous Namespace ဥပမာ

```
#include <iostream>

namespace {
    // ဤ function ကို ဤ file တွင်သာ အသုံးပြုနိုင်မည်
    void internalLog(const std::string& message) {
        std::cout << "[LOG] " << message << std::endl;
    }
}

namespace App {
    void start() {
        internalLog("Application starting");
        std::cout << "Running application..." << std::endl;
        internalLog("Application started successfully");
    }
}

int main() {
    App::start();
```

```
    // internalLog("Test"); // Error - anonymous namespace မှ function ကို အပြင်မှ မ
မြင်နိုင်
    return 0;
}
```

✓ ရှင်းလင်းချက်

- Anonymous namespace သည် file scope အတွက်သာ သက်ရောက်ပါမယ်။
- C++ တွင် static function များအစား အသုံးပြုနိုင်သော နည်းလမ်းတစ်ခုဖြစ်ပါတယ်။

5. Namespace Alias ဥပမာ

```
#include <iostream>

namespace VeryLongAndComplexNamespaceName {
    void importantFunction() {
        std::cout << "Important operation" << std::endl;
    }
}

// Namespace alias ဖန်တီးခြင်း
namespace Short = VeryLongAndComplexNamespaceName;

int main() {
    VeryLongAndComplexNamespaceName::importantFunction();
    Short::importantFunction();  // Alias ကိုသုံးပြီး ခေါ်ယူခြင်း

    return 0;
}
```

✓ ရှင်းလင်းချက်

- Namespace အမည်ရှည်ကြီးများအတွက် အမည်ပြောင်းသတ်မှတ်နိုင်ပါတယ်။
- ကုဒ်ဖတ်ရှုရလွယ်ကူစေရန် အထောက်အကူပြုပါတယ်။

Namespace များကို အသုံးပြုခြင်းဖြင့် သင့်၏ကုဒ်များသည် ပိုမိုစနစ်ကျပြီး ထိန်းသိမ်းရလွယ်ကူလာမှာ မည်ဖြစ်ပါတယ်။ Large-scale project များတွင် namespace များကို မဖြစ်မနေအသုံးပြုသင့်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန်

❒ City Express Store Point of Sale System

အောက်တွင် namespace, class နှင့် file များခွဲထားသော City Express Store အတွက် Point of Sale (POS) system ကုဒ်ကို တင်ပြထားပါသည်။

File Structure

```
CityExpressPOS/
├─── include/
│    ├─── Product.h
│    ├─── Inventory.h
│    ├─── Transaction.h
│    └─── POSSystem.h
├─── src/
│    ├─── Product.cpp
│    ├─── Inventory.cpp
│    ├─── Transaction.cpp
│    └─── POSSystem.cpp
└─── main.cpp
```

1. Product.h (Header File)

```
#ifndef PRODUCT_H
#define PRODUCT_H

#include <string>

namespace CityExpress {
    class Product {
    private:
        std::string id;
        std::string name;
        double price;
        int stock;

    public:
```

```
        Product(const std::string& id, const std::string& name, double price,
int stock);

        // Getter methods
        std::string getId() const;
        std::string getName() const;
        double getPrice() const;
        int getStock() const;

        // Setter methods
        void setPrice(double newPrice);
        void updateStock(int quantity);

        void display() const;
    };
}

#endif
```

2. Product.cpp (Implementation)

```
#include "Product.h"
#include <iostream>

namespace CityExpress {
    Product::Product(const std::string& id, const std::string& name, double p
rice, int stock)
        : id(id), name(name), price(price), stock(stock) {}

    std::string Product::getId() const { return id; }
    std::string Product::getName() const { return name; }
    double Product::getPrice() const { return price; }
    int Product::getStock() const { return stock; }

    void Product::setPrice(double newPrice) { price = newPrice; }

    void Product::updateStock(int quantity) {
        stock += quantity;
        if(stock < 0) stock = 0;
    }
```

```
    void Product::display() const {
        std::cout << "ID: " << id << " | Name: " << name
                  << " | Price: $" << price << " | Stock: " << stock << std::
endl;
    }
}
```

3. Inventory.h (Header File)

```
#ifndef INVENTORY_H
#define INVENTORY_H

#include <vector>
#include "Product.h"

namespace CityExpress {
    class Inventory {
    private:
        std::vector<Product> products;

    public:
        void addProduct(const Product& product);
        Product* findProduct(const std::string& id);
        void displayAllProducts() const;
        void updateProductStock(const std::string& id, int quantity);
    };
}

#endif
```

4. Inventory.cpp (Implementation)

```
#include "Inventory.h"
#include <algorithm>

namespace CityExpress {
    void Inventory::addProduct(const Product& product) {
        products.push_back(product);
    }
```

```
    Product* Inventory::findProduct(const std::string& id) {
        auto it = std::find_if(products.begin(), products.end(),
            [&id](const Product& p) { return p.getId() == id; });

        if(it != products.end()) return &(*it);
        return nullptr;
    }

    void Inventory::displayAllProducts() const {
        for(const auto& product : products) {
            product.display();
        }
    }

    void Inventory::updateProductStock(const std::string& id, int quantity) {
        Product* product = findProduct(id);
        if(product) product->updateStock(quantity);
    }
}
```

5. Transaction.h (Header File)

```
#ifndef TRANSACTION_H
#define TRANSACTION_H

#include <vector>
#include <ctime>
#include "Product.h"

namespace CityExpress {
    class Transaction {
    private:
        std::string transactionId;
        std::time_t dateTime;
        std::vector<std::pair<Product*, int>> items;
        double total;

    public:
        Transaction();
        void addItem(Product* product, int quantity);
```

```
        void calculateTotal();
        void displayReceipt() const;

        std::string getTransactionId() const { return transactionId; }
        double getTotal() const { return total; }
    };
}

#endif
```

6. Transaction.cpp (Implementation)

```
#include "Transaction.h"
#include <iomanip>
#include <sstream>
#include <ctime>

namespace CityExpress {
    Transaction::Transaction() {
        std::time(&dateTime);

        // Generate transaction ID (YYYYMMDD-HHMMSS)
        std::tm* tm = std::localtime(&dateTime);
        std::ostringstream oss;
        oss << std::put_time(tm, "%Y%m%d-%H%M%S");
        transactionId = oss.str();
        total = 0.0;
    }

    void Transaction::addItem(Product* product, int quantity) {
        if(product && quantity > 0 && product->getStock() >= quantity) {
            items.emplace_back(product, quantity);
            product->updateStock(-quantity);
            calculateTotal();
        }
    }

    void Transaction::calculateTotal() {
        total = 0.0;
        for(const auto& item : items) {
            total += item.first->getPrice() * item.second;
```

```
        }
    }

    void Transaction::displayReceipt() const {
        std::cout << "\n=== City Express Store ===" << std::endl;
        std::cout << "Transaction ID: " << transactionId << std::endl;
        std::cout << "Date: " << std::ctime(&dateTime);
        std::cout << "-------------------------" << std::endl;

        for(const auto& item : items) {
            std::cout << item.first->getName() << " x" << item.second
                      << " @ $" << item.first->getPrice()
                      << " = $" << (item.first->getPrice() * item.second) <<
std::endl;
        }

        std::cout << "-------------------------" << std::endl;
        std::cout << "TOTAL: $" << total << std::endl;
        std::cout << "Thank you for shopping with us!\n" << std::endl;
    }
}
```

7. POSSystem.h (Header File)

```
#ifndef POSSYSTEM_H
#define POSSYSTEM_H

#include "Inventory.h"
#include "Transaction.h"

namespace CityExpress {
    class POSSystem {
    private:
        Inventory inventory;

    public:
        void initializeSampleData();
        void run();
        void showMenu() const;
        void processSale();
```

```
        void addNewProduct();
        void viewInventory() const;
    };
}

#endif
```

8. POSSystem.cpp (Implementation)

```
#include "POSSystem.h"
#include <iostream>
#include <limits>

namespace CityExpress {
    void POSSystem::initializeSampleData() {
        inventory.addProduct(Product("P001", "Mineral Water", 1.50, 100));
        inventory.addProduct(Product("P002", "Bread", 3.20, 50));
        inventory.addProduct(Product("P003", "Milk", 2.80, 75));
    }

    void POSSystem::run() {
        initializeSampleData();
        int choice;

        do {
            showMenu();
            std::cout << "Enter your choice: ";
            std::cin >> choice;

            switch(choice) {
                case 1: processSale(); break;
                case 2: addNewProduct(); break;
                case 3: viewInventory(); break;
                case 4: std::cout << "Exiting system...\n"; break;
                default: std::cout << "Invalid choice. Try again.\n";
            }

            // Clear input buffer
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'
);
        } while(choice != 4);
```

```
}

void POSSystem::showMenu() const {
    std::cout << "\n=== City Express POS System ===" << std::endl;
    std::cout << "1. Process Sale" << std::endl;
    std::cout << "2. Add New Product" << std::endl;
    std::cout << "3. View Inventory" << std::endl;
    std::cout << "4. Exit" << std::endl;
}

void POSSystem::processSale() {
    Transaction transaction;
    std::string productId;
    int quantity;
    char moreItems;

    do {
        viewInventory();
        std::cout << "Enter product ID: ";
        std::cin >> productId;

        Product* product = inventory.findProduct(productId);
        if(!product) {
            std::cout << "Product not found!\n";
            continue;
        }

        std::cout << "Enter quantity: ";
        std::cin >> quantity;

        if(quantity <= 0 || quantity > product->getStock()) {
            std::cout << "Invalid quantity!\n";
            continue;
        }

        transaction.addItem(product, quantity);

        std::cout << "Add more items? (y/n): ";
        std::cin >> moreItems;
    } while(moreItems == 'y' || moreItems == 'Y');

    transaction.displayReceipt();
```

```
    }

    void POSSystem::addNewProduct() {
        std::string id, name;
        double price;
        int stock;

        std::cout << "Enter product ID: ";
        std::cin >> id;

        if(inventory.findProduct(id)) {
            std::cout << "Product ID already exists!\n";
            return;
        }

        std::cin.ignore();
        std::cout << "Enter product name: ";
        std::getline(std::cin, name);

        std::cout << "Enter price: ";
        std::cin >> price;

        std::cout << "Enter initial stock: ";
        std::cin >> stock;

        inventory.addProduct(Product(id, name, price, stock));
        std::cout << "Product added successfully!\n";
    }

    void POSSystem::viewInventory() const {
        std::cout << "\n=== Current Inventory ===" << std::endl;
        inventory.displayAllProducts();
    }
}
```

9. main.cpp (Application Entry Point)

```
#include "POSSystem.h"

int main() {
    CityExpress::POSSystem posSystem;
    posSystem.run();
    return 0;
}
```

✓ ရှင်းလင်းချက်

1. Namespace Structure
    - CityExpress namespace အတွင်းတွင် POS system ၏ class များအားလုံးကို စုစည်းထားပါသည်
    - နာမည်ပဋိပက္ခများကို ရှောင်ရှားနိုင်ရန် နှင့် code organization ကောင်းမွန်စေရန်

2. Class Design
    - Product: ကုန်ပစ္စည်းများ၏ အချက်အလက်များကို စီမံခန့်ခွဲမည်
    - Inventory: ကုန်ပစ္စည်းစာရင်းနှင့် stock level များကို စီမံမည်
    - Transaction: ရောင်းချမှုလုပ်ငန်းစဉ်နှင့် receipt ထုတ်ခြင်းကို ကိုင်တွယ်မည်
    - POSSystem: အဓိက application logic နှင့် user interface ကို စီမံမည်

3. File Organization
    - Header files (.h) များကို include folder တွင်ထားပါသည်
    - Implementation files (.cpp) များကို src folder တွင်ထားပါသည်
    - main.cpp သည် application ၏ entry point ဖြစ်ပါသည်

4. Features
    - ကုန်ပစ္စည်းများစီမံခန့်ခွဲခြင်း
    - ရောင်းချမှုလုပ်ငန်းစဉ်များ

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

- Inventory tracking
- Receipt generation
- Menu-based user interface

5. အသုံးပြုပုံ

- Program ကို run သောအခါ menu-based interface ပေါ်လာမည်
- User သည် sale ပြုလုပ်ခြင်း၊ ကုန်ပစ္စည်းအသစ်ထည့်ခြင်း၊ inventory ကြည့်ခြင်းများ ပြုလုပ်နိုင်မည်
- Sample data 3မျိုးဖြင့် အစပြုထားပါသည်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Thread

❒ Thread

C++ မှာ thread ဆိုတာ program တစ်ခုရဲ့ execution flow တစ်ခုဖြစ်ပြီး multiple threads တွေကို တစ်ပြိုင်နက် run နိုင်ပါတယ်။ C++11 မှာ <thread> header နဲ့ thread support ကို standard library အဖြစ် ထည့်သွင်းပေးထားပါတယ်။

✓ Thread အခြေခံအသုံးပြုနည်း

```
#include <iostream>
#include <thread>
using namespace std;

void threadFunction() {
    cout << "Thread မှ နှုတ်ဆက်ခြင်း!" << endl;
}

int main() {
    // Thread ဖန်တီးခြင်း
    thread t(threadFunction);

    cout << "Main function မှ နှုတ်ဆက်ခြင်း!" << endl;

    // Thread ပြီးဆုံးရန် စောင့်ခြင်း
    t.join();

    return 0;
}
```

✓ Thread များနှင့် Parameters များ

```
#include <iostream>
#include <thread>
#include <string>
using namespace std;

void greet(string name, int age) {
    cout << "မင်္ဂလာပါ " << name << "! သင့်အသက်က " << age << " နှစ်ပါ။" << endl;
}

int main() {
    thread t(greet, "ကျော်ကျော်", 25);
    t.join();
    return 0;
}
```

✓ Thread Management နည်းလမ်းများ

1. join() - Main thread က thread ပြီးဆုံးသည်အထိ စောင့်မည်
2. detach() - Thread ကို main thread မှ ခွဲထုတ်မည် (background တွင် run မည်)
3. joinable() - Thread က joinable ဖြစ်မဖြစ် စစ်ဆေးမည်

```
#include <iostream>
#include <thread>
#include <chrono>
using namespace std;

void countNumbers(int max) {
    for(int i=1; i<=max; i++) {
        cout << i << " ";
```

```
        this_thread::sleep_for(chrono::milliseconds(200));
    }
}

int main() {
    thread t(countNumbers, 10);

    if(t.joinable()) {
        t.detach(); // သို့မဟုတ် t.join();
    }

    // Main thread အလုပ်
    cout << "\nMain thread အလုပ်လုပ်နေပါတယ်..." << endl;

    // Detach လုပ်ထားလျှင် program မရပ်မီ စောင့်ရန်
    this_thread::sleep_for(chrono::seconds(3));

    return 0;
}
```

✓ Race Condition နှင့် Mutex

Threads များသည် shared data ကို တစ်ပြိုင်နက် access လုပ်ပါက race condition ဖြစ်နိုင်ပါတယ်။

```
#include <iostream>
#include <thread>
#include <mutex>
using namespace std;

mutex mtx; // Mutex object
int sharedData = 0;

void increment() {
    lock_guard<mutex> lock(mtx); // Automatic lock/unlock
    for(int i=0; i<100000; i++) {
        sharedData++;
    }
```

```
}

int main() {
    thread t1(increment);
    thread t2(increment);

    t1.join();
    t2.join();

    cout << "Final value: " << sharedData << endl; // 200000 ဖြစ်သင့်သည်

    return 0;
}
```

Thread အသုံးပြုရန် အကြံပြုချက်များ

1. RAII Principle ကို အသုံးပြုပါ (Resource Acquisition Is Initialization)
2. std::lock_guard သို့မဟုတ် std::unique_lock တို့ကို အသုံးပြုပါ
3. Deadlocks မဖြစ်အောင် သတိထားပါ
4. Thread-safe data structures များကို အသုံးပြုပါ

✓ Thread Pool Pattern (ဥပမာ)

```
#include <iostream>
#include <vector>
#include <thread>
#include <queue>
#include <mutex>
#include <condition_variable>
#include <functional>
using namespace std;

class ThreadPool {
```

```
public:
    ThreadPool(size_t numThreads) : stop(false) {
        for(size_t i=0; i<numThreads; ++i) {
            workers.emplace_back([this] {
                while(true) {
                    function<void()> task;
                    {
                        unique_lock<mutex> lock(queueMutex);
                        condition.wait(lock, [this] {
                            return stop || !tasks.empty();
                        });
                        if(stop && tasks.empty()) return;
                        task = move(tasks.front());
                        tasks.pop();
                    }
                    task();
                }
            });
        }
    }

    template<class F>
    void enqueue(F&& f) {
        {
            unique_lock<mutex> lock(queueMutex);
            tasks.emplace(forward<F>(f));
        }
        condition.notify_one();
    }

    ~ThreadPool() {
        {
            unique_lock<mutex> lock(queueMutex);
            stop = true;
        }
        condition.notify_all();
        for(thread& worker : workers) {
            worker.join();
        }
    }

private:
```

```
    vector<thread> workers;
    queue<function<void()>> tasks;
    mutex queueMutex;
    condition_variable condition;
    bool stop;
};

int main() {
    ThreadPool pool(4);

    for(int i=0; i<8; ++i) {
        pool.enqueue([i] {
            cout << "Task " << i << " ကို thread " << this_thread::get_id() <<
" မှလုပ်ဆောင်ပါတယ်\n";
            this_thread::sleep_for(chrono::seconds(1));
        });
    }

    return 0;
}
```

✓ မှတ်ချက်:

C++ threading ကို အသုံးပြုရာတွင် <thread>, <mutex>, <atomic>, <future>, <condition_variable> စသည့် headers များကို လေ့လာ သင့်ပါတယ်။ Threads များကို မှန်ကန်စွာ manage မလုပ်ပါက resource leaks နှင့် undefined behaviors များ ဖြစ်ပေါ်နိုင်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ လေ့ကျင့်ရန်

❒ Dining Philosophers Problem Application

Dining Philosophers Problem ဆိုတာ classical synchronization problem တစ်ခုဖြစ်ပြီး thread synchronization နဲ့ deadlock avoidance ကို လေ့လာဖို့ ကောင်းမွန်တဲ့ ဥပမာတစ်ခုပါ။

❒ Problem Definition

ဒီ problem မှာ philosopher 5 ယောက်ဟာ စားပွဲဝိုင်းတစ်ခုမှာ ထိုင်နေကြပြီး သူတို့ရှေ့မှာ ခြင်းတောင်းတစ်ခုစီနဲ့ ညာဘက်မှာ ခရင်းတစ်ချောင်းစီ ရှိပါတယ်။ Philosopher တစ်ယောက်ဟာ:

1. စဉ်းစားခြင်း (think)
2. ဗိုက်ဆာလာခြင်း (get hungry)
3. ခရင်းနှစ်ချောင်းကို ကောက်ယူခြင်း (pick up chopsticks)
4. စားခြင်း (eat)
5. ခရင်းများကို ပြန်ချခြင်း (put down chopsticks)

✓ Solution Implementation

```
#include <iostream>
#include <thread>
#include <mutex>
#include <vector>
#include <chrono>
#include <random>
using namespace std;
```

```
const int NUM_PHILOSOPHERS = 5;

mutex output_mtx; // Console output synchronization
mutex chopsticks[NUM_PHILOSOPHERS]; // Each chopstick has its own mutex

// Random number generator for varying think/eat times
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<> dis(100, 1000);

void philosopher(int id) {
    int left = id;
    int right = (id + 1) % NUM_PHILOSOPHERS;

    while(true) {
        // Thinking
        {
            lock_guard<mutex> lock(output_mtx);
            cout << "Philosopher " << id << " စဉ်းစားနေပါတယ်..." << endl;
        }
        this_thread::sleep_for(chrono::milliseconds(dis(gen)));

        // Hungry - Try to pick up chopsticks
        {
            lock_guard<mutex> lock(output_mtx);
            cout << "Philosopher " << id << " ဗိုက်ဆာနေပါပြီ..." << endl;
        }

        // Deadlock ကို ရှောင်ရန် odd-numbered philosophers များက ဘယ်ခရင်းကို အရင် ကောက်
မည်
        if(id % 2 == 0) {
            chopsticks[left].lock();
            {
                lock_guard<mutex> lock(output_mtx);
                cout << "Philosopher " << id << " ဘယ်ခရင်းကို ကောက်ယူပါပြီ" << endl
;
            }

            chopsticks[right].lock();
            {
```

```
                lock_guard<mutex> lock(output_mtx);
                cout << "Philosopher " << id << " ညာခရင်းကို ကောက်ယူပါပြီ" << endl;
            }
        } else {
            chopsticks[right].lock();
            {
                lock_guard<mutex> lock(output_mtx);
                cout << "Philosopher " << id << " ညာခရင်းကို ကောက်ယူပါပြီ" << endl;
            }

            chopsticks[left].lock();
            {
                lock_guard<mutex> lock(output_mtx);
                cout << "Philosopher " << id << " ဘယ်ခရင်းကို ကောက်ယူပါပြီ" << endl
;
            }
        }

        // Eating
        {
            lock_guard<mutex> lock(output_mtx);
            cout << "Philosopher " << id << " စားနေပါပြီ..." << endl;
        }
        this_thread::sleep_for(chrono::milliseconds(dis(gen)));

        // Put down chopsticks
        chopsticks[left].unlock();
        chopsticks[right].unlock();

        {
            lock_guard<mutex> lock(output_mtx);
            cout << "Philosopher " << id << " ခရင်းများကို ပြန်ချပါပြီ" << endl;
        }
    }
}

int main() {
    cout << "Dining Philosophers Problem Simulation\n";
    cout << "-----------------------------------\n";

    vector<thread> philosophers;
```

```
    // Create philosopher threads
    for(int i=0; i<NUM_PHILOSOPHERS; ++i) {
        philosophers.emplace_back(philosopher, i);
    }

    // Wait for all threads (though they run indefinitely)
    for(auto& ph : philosophers) {
        ph.join();
    }

    return 0;
}
```

✓ အလုပ်လုပ်ပုံ အဆင့်ဆင့်

1. Initialization
    - Philosopher 5 ယောက်နဲ့ ခရင်း 5 ချောင်း ဖန်တီးပါတယ်
    - ခရင်းတစ်ချောင်းစီကို mutex တစ်ခုစီနဲ့ ကိုယ်စားပြုပါတယ်
2. Philosopher Behavior
    - စဉ်းစားခြင်း (think): random time ယူပါတယ်
    - ဗိုက်ဆာခြင်း (get hungry): ခရင်းများကို ကောက်ယူဖို့ ကြိုးစားပါတယ်
    - စားခြင်း (eat): ခရင်းနှစ်ချောင်းရရင် random time ယူစားပါတယ်
    - ခရင်းများ ပြန်ချခြင်း (release chopsticks)
3. Deadlock Avoidance Strategy
    - မဂဏန်း philosopher များ (0,2,4) က ဘယ်ခရင်းကို အရင်ကောက်ပါတယ်
    - ဂဏန်း philosopher များ (1,3) က ညာခရင်းကို အရင်ကောက်ပါတယ်
    - ဒီနည်းလမ်းက circular wait condition ကို ဖြတ်တောက်ပြီး deadlock မဖြစ်အောင် ကာကွယ်ပေးပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

✓ ရှင်းလင်းချက်

1. Mutex အသုံးပြုခြင်း
    - ခရင်းတစ်ချောင်းစီကို mutex တစ်ခုစီနဲ့ ကိုယ်စားပြုပါတယ်
    - lock() နဲ့ unlock() ကို အသုံးပြုပြီး ခရင်းကောက်ယူခြင်း/ပြန်ချခြင်းတွေ လုပ်ပါတယ်
2. Output Synchronization
    - Console output အတွက် အခြား mutex (output_mtx) တစ်ခုကို အသုံးပြုပါတယ်
    - Threads များကနေ console ကို တစ်ပြိုင်နက် ရေးသားတာမျိုး မဖြစ်အောင် ကာကွယ်ပါတယ်
3. Random Timing
    - random_device နဲ့ uniform_int_distribution တို့ကို အသုံးပြုပြီး think/eat time တွေကို random ဖြစ်အောင် လုပ်ပါတယ်
    - ဒါမှ real-world scenario နဲ့ ပိုတူမှာဖြစ်ပါတယ်
4. Deadlock Prevention
    - Philosopher များကို ခရင်းကောက်တဲ့ အစီအစဉ် မတူအောင် လုပ်ထားပါတယ်
    - ဒါမှ circular wait condition မဖြစ်မှာပါ

✓ ဥပမာ Output

```
Dining Philosophers Problem Simulation
---------------------------------
Philosopher 0 စဉ်းစားနေပါတယ်...
Philosopher 1 စဉ်းစားနေပါတယ်...
Philosopher 2 စဉ်းစားနေပါတယ်...
Philosopher 3 စဉ်းစားနေပါတယ်...
Philosopher 4 စဉ်းစားနေပါတယ်...
Philosopher 1 ဗိုက်ဆာနေပါပြီ...
Philosopher 1 ညာခရင်းကို ကောက်ယူပါပြီ
Philosopher 1 ဘယ်ခရင်းကို ကောက်ယူပါပြီ
```

```
Philosopher 1 စားနေပါပြီ...
Philosopher 0 ဗိုက်ဆာနေပါပြီ...
Philosopher 0 ဘယ်ခရင်းကို ကောက်ယူပါပြီ
Philosopher 0 ညာခရင်းကို ကောက်ယူပါပြီ
Philosopher 0 စားနေပါပြီ...
...
```

✓ အဆိုပြုချက်များ

1. Termination Condition
   Program က infinite loop ဖြစ်နေတာကြောင့် termination condition တစ်ခုထည့်သင့်ပါတယ် (ဥပမာ - အချိန်သတ်မှတ်ချက် သို့မဟုတ် အကြိမ်ရေသတ်မှတ်ချက်)
2. Resource Hierarchy Solution
   ခရင်းတွေကို numbering လုပ်ပြီး philosopher တိုင်းက နံပါတ်ငယ်တဲ့ ခရင်းကို အရင်ကောက်တဲ့နည်းလည်း အသုံးပြုနိုင်ပါတယ်
3. std::unique_lock အသုံးပြုခြင်း
   lock_guard အစား unique_lock ကို အသုံးပြုပြီး ပိုမိုတောင့်တင်းခိုင်မာတဲ့ code ရေးနိုင်ပါတယ်
4. Timeout Mechanism
   try_lock နဲ့ timeout mechanism တွေ ထည့်သွင်းပြီး ပိုကောင်းတဲ့ solution တစ်ခု ဖန်တီးနိုင်ပါတယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

GAMES

❒ TIC TAC TOE GAME

အောက်မှာ C++ ဖြင့်ရေးထားသော Tic Tac Toe (XOX) ဂိမ်းကုဒ်နှင့် အသေးစိတ်ရှင်းလင်းချက်ကို တင်ပြထားပါတယ်။

```
#include <iostream>
#include <vector>

using namespace std;

// ဂိမ်းဘုတ်ကို ပြသမယ့် function
void displayBoard(const vector<vector<char>>& board) {
    cout << "\n";
    for (int i = 0; i < 3; i++) {
        cout << " ";
        for (int j = 0; j < 3; j++) {
            cout << board[i][j];
            if (j < 2) cout << " | ";
        }
        cout << "\n";
        if (i < 2) cout << "-----------\n";
    }
    cout << "\n";
}

// အနိုင်ရှိမရှိ စစ်ဆေးမယ့် function
bool checkWin(const vector<vector<char>>& board, char player) {
    // အတန်း၊ အကော်လံနှင့် ထောင့်ဖြတ်များကို စစ်ဆေးမယ်
    for (int i = 0; i < 3; i++) {
        // အတန်းများကို စစ်ဆေးမယ်
```

```
        if (board[i][0] == player && board[i][1] == player && board[i][2] ==
player)
            return true;
        // အကော်လံများကို စစ်ဆေးမယ်
        if (board[0][i] == player && board[1][i] == player && board[2][i] ==
player)
            return true;
    }
    // ထောင့်ဖြတ်များကို စစ်ဆေးမယ်
    if (board[0][0] == player && board[1][1] == player && board[2][2] == play
er)
        return true;
    if (board[0][2] == player && board[1][1] == player && board[2][0] == play
er)
        return true;
    return false;
}

// ဘုတ်ပြည့်နေပြီလားဆိုတာ စစ်ဆေးမယ့် function
bool checkDraw(const vector<vector<char>>& board) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ')
                return false;
        }
    }
    return true;
}

int main() {
    vector<vector<char>> board(3, vector<char>(3, ' ')); // 3x3 ဘုတ်ကို ဖန်တီးမယ်
    char currentPlayer = 'X'; // ပထမဆုံး player က X ဖြစ်မယ်
    bool gameOver = false;

    cout << "Tic Tac Toe ဂိမ်းကို ကြိုဆိုပါတယ်!\n";
    cout << "Player 1 (X) - Player 2 (O)\n";

    while (!gameOver) {
        displayBoard(board);
        int row, col;
```

```
        // လက်ရှိ player အတွက် input တောင်းမယ်
        cout << "Player " << currentPlayer << " ရဲ့အလှည့်\n";
        cout << "အတန်းနံပါတ် (1-3) ထည့်ပါ: ";
        cin >> row;
        cout << "အကော်လံနံပါတ် (1-3) ထည့်ပါ: ";
        cin >> col;

        // array index အတွက် 1 နှုတ်မယ်
        row--;
        col--;

        // input မှန်မမှန်စစ်ဆေးမယ်
        if (row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] ==
' ') {
            board[row][col] = currentPlayer;

            // အနိုင်ရှိမရှိစစ်ဆေးမယ်
            if (checkWin(board, currentPlayer)) {
                displayBoard(board);
                cout << "Player " << currentPlayer << " အနိုင်ရပါပြီ! ဂုဏ်ယူပါတယ်!\n"
;
                gameOver = true;
            }
            // ဂိမ်းအနှိုင်းအမျှတ်မရှိဘဲ ပြည့်နေပြီလားစစ်ဆေးမယ်
            else if (checkDraw(board)) {
                displayBoard(board);
                cout << "ဂိမ်းသရေကျပါပြီ! အနှိုင်းအမျှတ်မရှိပါ။\n";
                gameOver = true;
            }
            // player အလှည့်ပြောင်းမယ်
            else {
                currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
            }
        } else {
            cout << "မသင့်တော်သော input ဖြစ်နေပါသည်။ နေရာလွတ်ဖြစ်ပြီး 1-3 အတွင်းရှိနံပါတ်များ
ကိုသာ ထည့်ပါ။\n";
        }
    }
```

```
    return 0;
}
```

✓ ရှင်းလင်းချက်

1. ဘုတ်ပြသခြင်း (displayBoard function)
    - 3x3 matrix ကို လှပစွာပြသပေးမယ်
    - အတန်းတိုင်းတွင် "|" ဖြင့် ခွဲထားမယ်
    - အတန်းများကြားတွင် "----------" မျဉ်းဖြင့် ခွဲထားမယ်
2. အနိုင်စစ်ဆေးခြင်း (checkWin function)
    - အတန်းတူ 3ခု (ဥပမာ - ပထမအတန်းတွင် X X X)
    - အကော်လံတူ 3ခု (ဥပမာ - ပထမအကော်လံတွင် O O O)
    - ထောင့်ဖြတ်တူ 3ခု စစ်ဆေးမယ်
    - တစ်ခုခုနှင့်ကိုက်ညီရင် true ပြန်မယ်
3. သရေကျစစ်ဆေးခြင်း (checkDraw function)
    - ဘုတ်ပေါ်တွင် နေရာလွတ်မရှိတော့ရင် (အကုန် X နှင့် O ဖြင့်ပြည့်နေရင်) true ပြန်မယ်
4. အဓိက ဂိမ်းလုပ်ဆောင်ချက် (main function)
    - 3x3 ဘုတ်ကို စတင်ဖန်တီးမယ် (နေရာလွတ်များကို ' ' ဖြင့်ဖြည့်မယ်)
    - Player X ကို စတင်မယ်
    - ဂိမ်းမပြီးမချင်း loop ပတ်မယ်
    - Player ထံမှ အတန်း/အကော်လံနံပါတ်တောင်းမယ် (1-3)
    - Input မှန်ကန်မှုကို စစ်ဆေးမယ်
    - အနိုင်ရှိမရှိ (သို့) သရေကျမကျ စစ်ဆေးမယ်
    - အခြေအနေအလိုက် ရလဒ်ကို ပြသမယ်
    - Player အလှည့်ပြောင်းမယ်

✓ ဂိမ်းကိုဘယ်လို ကစားရမှာလဲ?

1. Player 1 (X) နှင့် Player 2 (O) အဖြစ် ကစားမယ်
2. အလှည့်ကျတိုင်း အတန်းနံပါတ် (1-3) နှင့် အကော်လံနံပါတ် (1-3) ထည့်ပါ
3. တစ်ယောက်ယောက်က တန်းလိုက်၊ ပြင်လိုက် (သို့) ထောင့်ဖြတ်အတိုင်း 3ခုတူရင် အနိုင်ရမယ်
4. ဘုတ်ပြည့်သွားပြီး အနိုင်အရှုံးမရှိရင် သရေကျမယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ SNAKE GAME

အောက်မှာ C++ ဖြင့်ရေးထားသော Snake Game ကုဒ်နှင့် အသေးစိတ်ရှင်းလင်းချက်ကို တင်ပြထားပါတယ်။

```
#include <iostream>
#include <conio.h>  // _kbhit() နှင့် _getch() အတွက်
#include <windows.h> // Sleep() အတွက်
#include <vector>
#include <cstdlib>  // rand() အတွက်
#include <ctime>    // time() အတွက်

using namespace std;

bool gameOver;
const int width = 20;
const int height = 20;
int x, y, fruitX, fruitY, score;
vector<pair<int, int>> snake; // မြွေကိုယ်အစိတ်အပိုင်းများ (x,y)
enum eDirection { STOP = 0, LEFT, RIGHT, UP, DOWN };
eDirection dir;

void Setup() {
    gameOver = false;
    dir = STOP;
    x = width / 2;
    y = height / 2;
    snake.clear();
```

```
    snake.push_back({x, y}); // မြွေခေါင်းကို အစပိုင်းထားမယ်
    srand(time(0));
    fruitX = rand() % width;
    fruitY = rand() % height;
    score = 0;
}

void Draw() {
    system("cls"); // console ကို ရှင်းမယ်

    // အပေါ်ဘောင်ရိုက်မယ်
    for (int i = 0; i < width + 2; i++)
        cout << "#";
    cout << endl;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (j == 0) cout << "#"; // ဘယ်ဘောင်ရိုက်မယ်

            // မြွေခေါင်းရှိရင် 'O' ပြမယ်
            if (i == y && j == x)
                cout << "O";
            // အသီးရှိရင် 'F' ပြမယ်
            else if (i == fruitY && j == fruitX)
                cout << "F";
            else {
                bool isBody = false;
                // မြွေကိုယ်အစိတ်အပိုင်းရှိရင် 'o' ပြမယ်
                for (auto segment : snake) {
                    if (segment.first == j && segment.second == i) {
                        cout << "o";
                        isBody = true;
                        break;
                    }
                }
                if (!isBody) cout << " ";
            }

            if (j == width - 1) cout << "#"; // ညာဘောင်ရိုက်မယ်
        }
```

```
        cout << endl;
    }

    // အောက်ဘောင်ရိုက်မယ်
    for (int i = 0; i < width + 2; i++)
        cout << "#";
    cout << endl;

    cout << "Score: " << score << endl;
}

void Input() {
    if (_kbhit()) {
        switch (_getch()) {
        case 'a':
            if (dir != RIGHT) dir = LEFT;
            break;
        case 'd':
            if (dir != LEFT) dir = RIGHT;
            break;
        case 'w':
            if (dir != DOWN) dir = UP;
            break;
        case 's':
            if (dir != UP) dir = DOWN;
            break;
        case 'x':
            case 'X':
            case 27: // ESC key
                gameOver = true;
                break;
        }
    }
}

void Logic() {
    pair<int, int> prev = {x, y};
    pair<int, int> prev2;

    // မြွေခေါင်းရွေ့မယ်
    switch (dir) {
    case LEFT:
```

```
        x--;
        break;
    case RIGHT:
        x++;
        break;
    case UP:
        y--;
        break;
    case DOWN:
        y++;
        break;
    }

    // နံရံနှင့်ထိရင် ဂိမ်းပြီးမယ်
    if (x >= width || x < 0 || y >= height || y < 0)
        gameOver = true;

    // ကိုယ်နှင့်ထိရင် ဂိမ်းပြီးမယ်
    for (auto segment : snake) {
        if (segment.first == x && segment.second == y) {
            gameOver = true;
            break;
        }
    }

    // အသီးကိုစားရင်
    if (x == fruitX && y == fruitY) {
        score += 10;
        fruitX = rand() % width;
        fruitY = rand() % height;
        snake.push_back({-1, -1}); // မြွေအလျားတိုးမယ်
    }

    // မြွေကိုယ်အစိတ်အပိုင်းများကို ရွှေ့မယ်
    for (int i = 0; i < snake.size(); i++) {
        if (i == 0) {
            prev2 = snake[i];
            snake[i] = prev;
        } else {
            pair<int, int> temp = snake[i];
            snake[i] = prev2;
```

```
            prev2 = temp;
        }
    }
}

int main() {
    Setup();
    while (!gameOver) {
        Draw();
        Input();
        Logic();
        Sleep(100); // ဂိမ်းအရှိန်ထိန်းဖို့
    }
    cout << "Game Over! Final Score: " << score << endl;
    return 0;
}
```

✓ ရှင်းလင်းချက်

1. အစပြုခြင်း (Setup function)
    - ဂိမ်းအခြေအနေများကို စတင်မယ်
    - မြွေခေါင်းကို အလယ်မှာထားမယ်
    - အသီးကို ကျပန်းနေရာတစ်ခုမှာထားမယ်
    - အချက်နှင့် မြွေအလျားကို သုည‌ထားမယ်
2. ပုံဆွဲခြင်း (Draw function)
    - console ကို ရှင်းမယ်
    - နံရံများကို '#' ဖြင့်ဆွဲမယ်
    - မြွေခေါင်းကို 'O'၊ ကိုယ်အစိတ်အပိုင်းများကို 'o'၊ အသီးကို 'F' ဖြင့်ပြမယ်
    - အချက်နှင့် နံရံများကို ပြမယ်
3. အသုံးပြုသူ input ဖတ်ခြင်း (Input function)
    - 'w', 'a', 's', 'd' ခလုတ်များဖြင့် မြွေလှည့်မယ်
    - ESC သို့မဟုတ် 'x' နှိပ်ရင် ဂိမ်းရပ်မယ်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

- မြွေကို ရုတ်တရက် ဆန့်ကျင်ဘက်သို့မလှည့်စေရန် ကာကွယ်မယ်

4. Logic function
   - မြွေခေါင်းကို ရွေ့မယ်
   - နံရံနှင့်ထိရင် ဂိမ်းပြီးမယ်
   - မြွေကိုယ်နှင့်ထိရင် ဂိမ်းပြီးမယ်
   - အသီးစားရင် အချက်တက်မယ်၊ မြွေအလျားတိုးမယ်
   - မြွေကိုယ်အစိတ်အပိုင်းများကို ရွေ့မယ်
5. **main function**
   - ဂိမ်းကို စတင်မယ်
   - ဂိမ်းမပြီးမချင်း loop ပတ်မယ်
   - ပုံဆွဲမယ်၊ input ဖတ်မယ်၊ ယန္တရားအလုပ်လုပ်မယ်
   - 100ms delay ထည့်ပေးမယ် (ဂိမ်းအရှိန်ထိန်းဖို့)
   - ဂိမ်းပြီးရင် အချက်ပြမယ်

✓ ဂိမ်းကစားနည်း

1. 'w' - အပေါ်သို့
2. 'a' - ဘယ်သို့
3. 's' - အောက်သို့
4. 'd' - ညာသို့
5. 'x' သို့မဟုတ် ESC - ဂိမ်းရပ်
6. အသီး (F) ကိုစားပြီး အချက်တင်မယ်
7. နံရံသို့မဟုတ် ကိုယ်နှင့်ထိရင် ဂိမ်းပြီးမယ်

ဤကုဒ်သည် console-based Snake Game တစ်ခုဖြစ်ပြီး Windows အတွက်သာ အလုပ်လုပ်ပါတယ်။ Linux/macOS တွင် အလုပ်လုပ်ရန် conio.h အစား ncurses library ကို အသုံးပြုရနိုင်ပါတယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

Practical Projects

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ Asia Royal Hospital Management System

```
asia_royal_hospital/
├── include/
│   ├── Patient.h
│   ├── Doctor.h
│   ├── Appointment.h
│   └── HospitalSystem.h
├── src/
│   ├── Patient.cpp
│   ├── Doctor.cpp
│   ├── Appointment.cpp
│   ├── HospitalSystem.cpp
│   └── main.cpp
├── data/
│   ├── patients.dat
│   ├── doctors.dat
│   └── appointments.dat
└── Makefile
```

✓ Header Files

1. Patient.h

```
#ifndef PATIENT_H
#define PATIENT_H

#include <string>
#include <iostream>

class Patient {
private:
    std::string id;
    std::string name;
    std::string phone;
    std::string address;
    std::string nrc;
    int age;
    std::string gender;

public:
    Patient(std::string i = "", std::string n = "", std::string p = "",
            std::string a = "", std::string nr = "", int ag = 0, std::string
g = "");

    // Getter methods
    std::string getId() const;
    std::string getName() const;
    std::string getPhone() const;
    std::string getAddress() const;
    std::string getNRC() const;
    int getAge() const;
    std::string getGender() const;

    void display() const;
};

#endif
```

2. Doctor.h

```
#ifndef DOCTOR_H
#define DOCTOR_H

#include <string>
#include <iostream>

class Doctor {
private:
    std::string id;
    std::string name;
    std::string specialty;
    std::string phone;
    std::string schedule;

public:
    Doctor(std::string i = "", std::string n = "", std::string s = "",
           std::string p = "", std::string sch = "");

    // Getter methods
    std::string getId() const;
    std::string getName() const;
    std::string getSpecialty() const;
    std::string getPhone() const;
    std::string getSchedule() const;

    void display() const;
};

#endif
```

3. Appointment.h

```
#ifndef APPOINTMENT_H
#define APPOINTMENT_H

#include <string>
```

```
#include <iostream>
#include <iomanip>

class Appointment {
private:
    std::string appointmentId;
    std::string patientId;
    std::string doctorId;
    std::string date;
    std::string time;
    std::string reason;
    bool isPaid;
    double fee;

public:
    Appointment(std::string aId = "", std::string pId = "", std::string dId =
"",
                std::string d = "", std::string t = "", std::string r = "",
                bool paid = false, double f = 0.0);

    // Getter methods
    std::string getAppointmentId() const;
    std::string getPatientId() const;
    std::string getDoctorId() const;
    std::string getDate() const;
    std::string getTime() const;
    std::string getReason() const;
    bool getIsPaid() const;
    double getFee() const;

    // Setter method
    void markAsPaid();

    void display() const;
};

#endif
```

4. HospitalSystem.h

```
#ifndef HOSPITAL_SYSTEM_H
#define HOSPITAL_SYSTEM_H

#include <vector>
#include <string>
#include <fstream>
#include "Patient.h"
#include "Doctor.h"
#include "Appointment.h"

class HospitalManagementSystem {
private:
    std::vector<Patient> patients;
    std::vector<Doctor> doctors;
    std::vector<Appointment> appointments;

    const std::string PATIENTS_FILE = "data/patients.dat";
    const std::string DOCTORS_FILE = "data/doctors.dat";
    const std::string APPOINTMENTS_FILE = "data/appointments.dat";

    void loadData();
    void saveData();
    std::string generateId(const std::string& prefix);

public:
    HospitalManagementSystem();
    ~HospitalManagementSystem();

    void registerPatient();
    void registerDoctor();
    void scheduleAppointment();
    void processPayment();
    void displayPatients() const;
    void displayDoctors() const;
    void displayAppointments() const;
};
```

```
#endif
```

✓ Source Files

1. Patient.cpp

```
#include "Patient.h"

Patient::Patient(std::string i, std::string n, std::string p,
                 std::string a, std::string nr, int ag, std::string g)
    : id(i), name(n), phone(p), address(a), nrc(nr), age(ag), gender(g) {}

std::string Patient::getId() const { return id; }
std::string Patient::getName() const { return name; }
std::string Patient::getPhone() const { return phone; }
std::string Patient::getAddress() const { return address; }
std::string Patient::getNRC() const { return nrc; }
int Patient::getAge() const { return age; }
std::string Patient::getGender() const { return gender; }

void Patient::display() const {
    std::cout << "\n=== Patient Information ===" << std::endl;
    std::cout << "ID: " << id << std::endl;
    std::cout << "Name: " << name << std::endl;
    std::cout << "Phone: " << phone << std::endl;
    std::cout << "Address: " << address << std::endl;
    std::cout << "NRC: " << nrc << std::endl;
    std::cout << "Age: " << age << std::endl;
    std::cout << "Gender: " << gender << std::endl;
}
```

2. Doctor.cpp

```
#include "Doctor.h"
```

```
Doctor::Doctor(std::string i, std::string n, std::string s,
               std::string p, std::string sch)
    : id(i), name(n), specialty(s), phone(p), schedule(sch) {}

std::string Doctor::getId() const { return id; }
std::string Doctor::getName() const { return name; }
std::string Doctor::getSpecialty() const { return specialty; }
std::string Doctor::getPhone() const { return phone; }
std::string Doctor::getSchedule() const { return schedule; }

void Doctor::display() const {
    std::cout << "\n=== Doctor Information ===" << std::endl;
    std::cout << "ID: " << id << std::endl;
    std::cout << "Name: Dr. " << name << std::endl;
    std::cout << "Specialty: " << specialty << std::endl;
    std::cout << "Phone: " << phone << std::endl;
    std::cout << "Schedule: " << schedule << std::endl;
}
```

3. Appointment.cpp

```
#include "Appointment.h"

Appointment::Appointment(std::string aId, std::string pId, std::string dId,
                         std::string d, std::string t, std::string r,
                         bool paid, double f)
    : appointmentId(aId), patientId(pId), doctorId(dId),
      date(d), time(t), reason(r), isPaid(paid), fee(f) {}

std::string Appointment::getAppointmentId() const { return appointmentId; }
std::string Appointment::getPatientId() const { return patientId; }
std::string Appointment::getDoctorId() const { return doctorId; }
std::string Appointment::getDate() const { return date; }
std::string Appointment::getTime() const { return time; }
std::string Appointment::getReason() const { return reason; }
bool Appointment::getIsPaid() const { return isPaid; }
double Appointment::getFee() const { return fee; }

void Appointment::markAsPaid() { isPaid = true; }
```

```
void Appointment::display() const {
    std::cout << "\n=== Appointment Details ===" << std::endl;
    std::cout << "Appointment ID: " << appointmentId << std::endl;
    std::cout << "Patient ID: " << patientId << std::endl;
    std::cout << "Doctor ID: " << doctorId << std::endl;
    std::cout << "Date: " << date << std::endl;
    std::cout << "Time: " << time << std::endl;
    std::cout << "Reason: " << reason << std::endl;
    std::cout << "Fee: $" << std::fixed << std::setprecision(2) << fee << std
::endl;
    std::cout << "Payment Status: " << (isPaid ? "Paid" : "Pending") << std::
endl;
}
```

4. HospitalSystem.cpp

```
#include "HospitalSystem.h"
#include <ctime>
#include <limits>

HospitalManagementSystem::HospitalManagementSystem() {
    loadData();
}

HospitalManagementSystem::~HospitalManagementSystem() {
    saveData();
}

void HospitalManagementSystem::loadData() {
    std::ifstream patientFile(PATIENTS_FILE, std::ios::binary);
    std::ifstream doctorFile(DOCTORS_FILE, std::ios::binary);
    std::ifstream appointmentFile(APPOINTMENTS_FILE, std::ios::binary);

    Patient tempPatient;
    while (patientFile.read(reinterpret_cast<char*>(&tempPatient), sizeof(Pat
ient))) {
        patients.push_back(tempPatient);
    }
```

```
    Doctor tempDoctor;
    while (doctorFile.read(reinterpret_cast<char*>(&tempDoctor), sizeof(Docto
r))) {
        doctors.push_back(tempDoctor);
    }

    Appointment tempAppointment;
    while (appointmentFile.read(reinterpret_cast<char*>(&tempAppointment), si
zeof(Appointment))) {
        appointments.push_back(tempAppointment);
    }
}

void HospitalManagementSystem::saveData() {
    std::ofstream patientFile(PATIENTS_FILE, std::ios::binary);
    std::ofstream doctorFile(DOCTORS_FILE, std::ios::binary);
    std::ofstream appointmentFile(APPOINTMENTS_FILE, std::ios::binary);

    for (const auto& patient : patients) {
        patientFile.write(reinterpret_cast<const char*>(&patient), sizeof(Pat
ient));
    }

    for (const auto& doctor : doctors) {
        doctorFile.write(reinterpret_cast<const char*>(&doctor), sizeof(Docto
r));
    }

    for (const auto& appointment : appointments) {
        appointmentFile.write(reinterpret_cast<const char*>(&appointment), si
zeof(Appointment));
    }
}

std::string HospitalManagementSystem::generateId(const std::string& prefix) {
    std::time_t now = std::time(nullptr);
    return prefix + std::to_string(now % 10000);
}

void HospitalManagementSystem::registerPatient() {
    std::string name, phone, address, nrc, gender;
    int age;
```

```
    std::cin.ignore();
    std::cout << "\n=== Patient Registration ===" << std::endl;
    std::cout << "Enter patient name: ";
    std::getline(std::cin, name);
    std::cout << "Enter phone number: ";
    std::getline(std::cin, phone);
    std::cout << "Enter address: ";
    std::getline(std::cin, address);
    std::cout << "Enter NRC number: ";
    std::getline(std::cin, nrc);
    std::cout << "Enter age: ";
    std::cin >> age;
    std::cin.ignore();
    std::cout << "Enter gender: ";
    std::getline(std::cin, gender);

    std::string id = generateId("PAT");
    patients.emplace_back(id, name, phone, address, nrc, age, gender);

    std::cout << "\nPatient registered successfully!" << std::endl;
    std::cout << "Patient ID: " << id << std::endl;
}

void HospitalManagementSystem::registerDoctor() {
    std::string name, specialty, phone, schedule;

    std::cin.ignore();
    std::cout << "\n=== Doctor Registration ===" << std::endl;
    std::cout << "Enter doctor name: ";
    std::getline(std::cin, name);
    std::cout << "Enter specialty: ";
    std::getline(std::cin, specialty);
    std::cout << "Enter phone number: ";
    std::getline(std::cin, phone);
    std::cout << "Enter schedule (e.g., Mon-Fri 9AM-5PM): ";
    std::getline(std::cin, schedule);

    std::string id = generateId("DOC");
    doctors.emplace_back(id, name, specialty, phone, schedule);

    std::cout << "\nDoctor registered successfully!" << std::endl;
```

```
    std::cout << "Doctor ID: " << id << std::endl;
}

void HospitalManagementSystem::scheduleAppointment() {
    std::string patientId, doctorId, date, time, reason;
    double fee;

    std::cout << "\n=== Schedule Appointment ===" << std::endl;

    std::cout << "\n--- Patients List ---" << std::endl;
    for (const auto& patient : patients) {
        std::cout << patient.getId() << " - " << patient.getName() << std::en
dl;
    }
    std::cout << "Enter patient ID: ";
    std::cin >> patientId;

    std::cout << "\n--- Doctors List ---" << std::endl;
    for (const auto& doctor : doctors) {
        std::cout << doctor.getId() << " - Dr. " << doctor.getName()
                  << " (" << doctor.getSpecialty() << ")" << std::endl;
    }
    std::cout << "Enter doctor ID: ";
    std::cin >> doctorId;

    std::cin.ignore();
    std::cout << "Enter appointment date (DD/MM/YYYY): ";
    std::getline(std::cin, date);
    std::cout << "Enter appointment time: ";
    std::getline(std::cin, time);
    std::cout << "Enter reason for appointment: ";
    std::getline(std::cin, reason);
    std::cout << "Enter consultation fee: $";
    std::cin >> fee;

    std::string appointmentId = generateId("APT");
    appointments.emplace_back(appointmentId, patientId, doctorId, date, time,
reason, false, fee);

    std::cout << "\nAppointment scheduled successfully!" << std::endl;
    std::cout << "Appointment ID: " << appointmentId << std::endl;
}
```

```
void HospitalManagementSystem::processPayment() {
    std::string appointmentId;
    std::cout << "\n=== Process Payment ===" << std::endl;
    std::cout << "Enter appointment ID: ";
    std::cin >> appointmentId;

    bool found = false;
    for (auto& appointment : appointments) {
        if (appointment.getAppointmentId() == appointmentId && !appointment.g
etIsPaid()) {
            found = true;

            appointment.display();

            char confirm;
            std::cout << "\nConfirm payment (Y/N)? ";
            std::cin >> confirm;

            if (toupper(confirm) == 'Y') {
                appointment.markAsPaid();
                std::cout << "Payment processed successfully!" << std::endl;
            } else {
                std::cout << "Payment cancelled." << std::endl;
            }
            break;
        }
    }

    if (!found) {
        std::cout << "Appointment not found or already paid." << std::endl;
    }
}

void HospitalManagementSystem::displayPatients() const {
    std::cout << "\n=== Patient List ===" << std::endl;
    if (patients.empty()) {
        std::cout << "No patients registered." << std::endl;
    } else {
        for (const auto& patient : patients) {
            patient.display();
            std::cout << "----------------------" << std::endl;
```

```
        }
    }
}

void HospitalManagementSystem::displayDoctors() const {
    std::cout << "\n=== Doctor List ===" << std::endl;
    if (doctors.empty()) {
        std::cout << "No doctors registered." << std::endl;
    } else {
        for (const auto& doctor : doctors) {
            doctor.display();
            std::cout << "----------------------" << std::endl;
        }
    }
}

void HospitalManagementSystem::displayAppointments() const {
    std::cout << "\n=== Appointment List ===" << std::endl;
    if (appointments.empty()) {
        std::cout << "No appointments scheduled." << std::endl;
    } else {
        for (const auto& appointment : appointments) {
            appointment.display();
            std::cout << "----------------------" << std::endl;
        }
    }
}
```

5. main.cpp

```
#include "HospitalSystem.h"
#include <iostream>

void displayMainMenu() {
    std::cout << "\n=== Asia Royal Hospital Management System ===" << std::en
dl;
    std::cout << "1. Register New Patient" << std::endl;
    std::cout << "2. Register New Doctor" << std::endl;
    std::cout << "3. Schedule Appointment" << std::endl;
    std::cout << "4. Process Payment" << std::endl;
```

```
    std::cout << "5. View Patients" << std::endl;
    std::cout << "6. View Doctors" << std::endl;
    std::cout << "7. View Appointments" << std::endl;
    std::cout << "8. Exit" << std::endl;
    std::cout << "Enter your choice: ";
}

int main() {
    HospitalManagementSystem hospitalSystem;
    int choice;

    do {
        displayMainMenu();
        std::cin >> choice;

        switch(choice) {
            case 1:
                hospitalSystem.registerPatient();
                break;
            case 2:
                hospitalSystem.registerDoctor();
                break;
            case 3:
                hospitalSystem.scheduleAppointment();
                break;
            case 4:
                hospitalSystem.processPayment();
                break;
            case 5:
                hospitalSystem.displayPatients();
                break;
            case 6:
                hospitalSystem.displayDoctors();
                break;
            case 7:
                hospitalSystem.displayAppointments();
                break;
            case 8:
                std::cout << "Exiting system. Data saved automatically." << s
td::endl;
                break;
            default:
```

```
                std::cout << "Invalid choice. Please try again." << std::endl
;
        }
    } while (choice != 8);

    return 0;
}
```

✓ Makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -Iinclude
LDFLAGS =
SRCDIR = src
OBJDIR = obj
BINDIR = bin

SOURCES = $(wildcard $(SRCDIR)/*.cpp)
OBJECTS = $(patsubst $(SRCDIR)/%.cpp,$(OBJDIR)/%.o,$(SOURCES))
EXECUTABLE = $(BINDIR)/hospital_system

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
        @mkdir -p $(BINDIR)
        $(CXX) $(LDFLAGS) $^ -o $@

$(OBJDIR)/%.o: $(SRCDIR)/%.cpp
        @mkdir -p $(OBJDIR)
        $(CXX) $(CXXFLAGS) -c $< -o $@

clean:
        rm -rf $(OBJDIR) $(BINDIR)

run: $(EXECUTABLE)
        ./$(EXECUTABLE)

.PHONY: all clean run
```

✓ Testing Results

Test Case 1: Patient Registration

```
=== Patient Registration ===
Enter patient name: John Doe
Enter phone number: 09123456789
Enter address: No.123, Yangon
Enter NRC number: 12/YGN(N)123456
Enter age: 30
Enter gender: Male

Patient registered successfully!
Patient ID: PAT8321
```

Test Case 2: Doctor Registration

```
=== Doctor Registration ===
Enter doctor name: Smith
Enter specialty: Cardiology
Enter phone number: 09234567890
Enter schedule: Mon-Fri 9AM-5PM

Doctor registered successfully!
Doctor ID: DOC8322
```

Test Case 3: Schedule Appointment

```
=== Schedule Appointment ===

--- Patients List ---
PAT8321 - John Doe
Enter patient ID: PAT8321
```

```
--- Doctors List ---
DOC8322 - Dr. Smith (Cardiology)
Enter doctor ID: DOC8322
Enter appointment date (DD/MM/YYYY): 15/06/2023
Enter appointment time: 10:00 AM
Enter reason for appointment: Heart checkup
Enter consultation fee: $50

Appointment scheduled successfully!
Appointment ID: APT8323
```

Test Case 4: Process Payment

```
=== Process Payment ===
Enter appointment ID: APT8323

=== Appointment Details ===
Appointment ID: APT8323
Patient ID: PAT8321
Doctor ID: DOC8322
Date: 15/06/2023
Time: 10:00 AM
Reason: Heart checkup
Fee: $50.00
Payment Status: Pending

Confirm payment (Y/N)? Y
Payment processed successfully!
```

Test Case 5: View Records

```
=== Patient List ===

=== Patient Information ===
ID: PAT8321
Name: John Doe
Phone: 09123456789
```

```
Address: No.123, Yangon
NRC: 12/YGN(N)123456
Age: 30
Gender: Male
---------------------

=== Doctor List ===

=== Doctor Information ===
ID: DOC8322
Name: Dr. Smith
Specialty: Cardiology
Phone: 09234567890
Schedule: Mon-Fri 9AM-5PM
---------------------

=== Appointment List ===

=== Appointment Details ===
Appointment ID: APT8323
Patient ID: PAT8321
Doctor ID: DOC8322
Date: 15/06/2023
Time: 10:00 AM
Reason: Heart checkup
Fee: $50.00
Payment Status: Paid
---------------------
```

❒ KBZ Bank ATM Application

```
KBZBankATM/
├─── include/
│    ├─── Account.h
│    ├─── Transaction.h
│    ├─── ATM.h
│    └─── Database.h
├─── src/
│    ├─── Account.cpp
│    ├─── Transaction.cpp
│    ├─── ATM.cpp
│    └─── Database.cpp
└─── main.cpp
```

1. Account.h (Header File)

```
#ifndef ACCOUNT_H
#define ACCOUNT_H

#include <string>

namespace KBZBank {
    class Account {
    private:
        std::string accountNumber;
```

```
        std::string pin;
        double balance;
        std::string accountHolder;

    public:
        Account(const std::string& accNum, const std::string& pin,
                double balance, const std::string& holder);

        // Getter methods
        std::string getAccountNumber() const;
        std::string getAccountHolder() const;
        double getBalance() const;

        // Account operations
        bool verifyPin(const std::string& inputPin) const;
        bool withdraw(double amount);
        bool deposit(double amount);
        bool transfer(Account& targetAccount, double amount);

        void displayAccountInfo() const;
    };
}

#endif
```

2. Account.cpp (Implementation)

```
#include "Account.h"
#include <iostream>

namespace KBZBank {
    Account::Account(const std::string& accNum, const std::string& pin,
                     double balance, const std::string& holder)
        : accountNumber(accNum), pin(pin), balance(balance), accountHolder(ho
lder) {}

    std::string Account::getAccountNumber() const { return accountNumber; }
    std::string Account::getAccountHolder() const { return accountHolder; }
    double Account::getBalance() const { return balance; }

    bool Account::verifyPin(const std::string& inputPin) const {
```

```
        return pin == inputPin;
    }

    bool Account::withdraw(double amount) {
        if(amount > 0 && amount <= balance) {
            balance -= amount;
            return true;
        }
        return false;
    }

    bool Account::deposit(double amount) {
        if(amount > 0) {
            balance += amount;
            return true;
        }
        return false;
    }

    bool Account::transfer(Account& targetAccount, double amount) {
        if(withdraw(amount)) {
            targetAccount.deposit(amount);
            return true;
        }
        return false;
    }

    void Account::displayAccountInfo() const {
        std::cout << "\n=== Account Information ===" << std::endl;
        std::cout << "Account Holder: " << accountHolder << std::endl;
        std::cout << "Account Number: " << accountNumber << std::endl;
        std::cout << "Current Balance: " << balance << " MMK" << std::endl;
    }
}
```

3. Transaction.h (Header File)

```
#ifndef TRANSACTION_H
#define TRANSACTION_H

#include <string>
```

```
#include <ctime>

namespace KBZBank {
    class Transaction {
    public:
        enum class Type { WITHDRAWAL, DEPOSIT, TRANSFER, BALANCE_INQUIRY };

    private:
        std::string transactionId;
        std::time_t timestamp;
        Type type;
        std::string accountNumber;
        double amount;
        std::string relatedAccount; // For transfer transactions

    public:
        Transaction(Type type, const std::string& accNum, double amount,
                    const std::string& relatedAcc = "");

        std::string getTransactionId() const;
        std::string getTypeString() const;
        void displayReceipt() const;
    };
}

#endif
```

4. Transaction.cpp (Implementation)

```
#include "Transaction.h"
#include <iomanip>
#include <sstream>
#include <ctime>

namespace KBZBank {
    Transaction::Transaction(Type type, const std::string& accNum,
                             double amount, const std::string& relatedAcc)
        : type(type), accountNumber(accNum), amount(amount), relatedAccount(r
elatedAcc) {
        std::time(&timestamp);
```

```
        // Generate transaction ID (YYYYMMDDHHMMSS)
        std::tm* tm = std::localtime(&timestamp);
        std::ostringstream oss;
        oss << std::put_time(tm, "%Y%m%d%H%M%S");
        transactionId = oss.str();
    }

    std::string Transaction::getTransactionId() const {
        return transactionId;
    }

    std::string Transaction::getTypeString() const {
        switch(type) {
            case Type::WITHDRAWAL: return "Withdrawal";
            case Type::DEPOSIT: return "Deposit";
            case Type::TRANSFER: return "Transfer";
            case Type::BALANCE_INQUIRY: return "Balance Inquiry";
            default: return "Unknown";
        }
    }

    void Transaction::displayReceipt() const {
        std::cout << "\n=== KBZ Bank Receipt ===" << std::endl;
        std::cout << "Transaction ID: " << transactionId << std::endl;
        std::cout << "Date: " << std::ctime(&timestamp);
        std::cout << "Account: " << accountNumber << std::endl;
        std::cout << "Transaction Type: " << getTypeString() << std::endl;

        if(type == Type::TRANSFER) {
            std::cout << "To Account: " << relatedAccount << std::endl;
        }

        if(type != Type::BALANCE_INQUIRY) {
            std::cout << "Amount: " << amount << " MMK" << std::endl;
        }

        std::cout << "Thank you for banking with KBZ!\n" << std::endl;
    }
}
```

5. Database.h (Header File)

```
#ifndef DATABASE_H
#define DATABASE_H

#include <vector>
#include "Account.h"

namespace KBZBank {
    class Database {
    private:
        std::vector<Account> accounts;

    public:
        Database();
        Account* findAccount(const std::string& accountNumber);
        void initializeSampleData();
    };
}

#endif
```

6. Database.cpp (Implementation)

```
#include "Database.h"

namespace KBZBank {
    Database::Database() {
        initializeSampleData();
    }

    Account* Database::findAccount(const std::string& accountNumber) {
        for(auto& account : accounts) {
            if(account.getAccountNumber() == accountNumber) {
                return &account;
            }
        }
        return nullptr;
    }

    void Database::initializeSampleData() {
        accounts.emplace_back("123456789", "1234", 500000.0, "U Ba");
```

```
        accounts.emplace_back("987654321", "4321", 1000000.0, "Daw Mya");
        accounts.emplace_back("112233445", "1111", 250000.0, "Maung Hla");
    }
}
```

7. ATM.h (Header File)

```
#ifndef ATM_H
#define ATM_H

#include "Database.h"
#include "Transaction.h"

namespace KBZBank {
    class ATM {
    private:
        Database database;
        Account* currentAccount;

        void showMainMenu() const;
        void showTransactionMenu() const;
        void processWithdrawal();
        void processDeposit();
        void processTransfer();
        void processBalanceInquiry();

    public:
        void run();
        bool authenticateUser();
    };
}

#endif
```

8. ATM.cpp (Implementation)

```
#include "ATM.h"
#include <iostream>
#include <limits>
```

```
namespace KBZBank {
    void ATM::run() {
        std::cout << "Welcome to KBZ Bank ATM\n" << std::endl;

        if(authenticateUser()) {
            int choice;
            do {
                showMainMenu();
                std::cout << "Enter your choice: ";
                std::cin >> choice;

                switch(choice) {
                    case 1: processWithdrawal(); break;
                    case 2: processDeposit(); break;
                    case 3: processTransfer(); break;
                    case 4: processBalanceInquiry(); break;
                    case 5: std::cout << "Logging out...\n"; break;
                    default: std::cout << "Invalid choice. Please try again.\
n";
                }

                // Clear input buffer
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            } while(choice != 5);
        }
    }

    bool ATM::authenticateUser() {
        std::string accountNumber, pin;

        std::cout << "Please enter your account number: ";
        std::cin >> accountNumber;

        currentAccount = database.findAccount(accountNumber);
        if(!currentAccount) {
            std::cout << "Account not found.\n";
            return false;
        }

        std::cout << "Enter your PIN: ";
```

```
        std::cin >> pin;

        if(!currentAccount->verifyPin(pin)) {
            std::cout << "Invalid PIN.\n";
            return false;
        }

        std::cout << "\nAuthentication successful. Welcome, "
                  << currentAccount->getAccountHolder() << "!\n";
        return true;
    }

    void ATM::showMainMenu() const {
        std::cout << "\n=== Main Menu ===" << std::endl;
        std::cout << "1. Withdraw Cash" << std::endl;
        std::cout << "2. Deposit Cash" << std::endl;
        std::cout << "3. Transfer Money" << std::endl;
        std::cout << "4. Balance Inquiry" << std::endl;
        std::cout << "5. Exit" << std::endl;
    }

    void ATM::processWithdrawal() {
        double amount;
        std::cout << "\nEnter amount to withdraw: ";
        std::cin >> amount;

        if(currentAccount->withdraw(amount)) {
            Transaction tx(Transaction::Type::WITHDRAWAL,
                           currentAccount->getAccountNumber(), amount);
            tx.displayReceipt();
        } else {
            std::cout << "Withdrawal failed. Insufficient balance or invalid
amount.\n";
        }
    }

    void ATM::processDeposit() {
        double amount;
        std::cout << "\nEnter amount to deposit: ";
        std::cin >> amount;

        if(currentAccount->deposit(amount)) {
```

```
            Transaction tx(Transaction::Type::DEPOSIT,
                           currentAccount->getAccountNumber(), amount);
            tx.displayReceipt();
        } else {
            std::cout << "Deposit failed. Invalid amount.\n";
        }
    }

    void ATM::processTransfer() {
        std::string targetAccountNumber;
        double amount;

        std::cout << "\nEnter recipient account number: ";
        std::cin >> targetAccountNumber;

        Account* targetAccount = database.findAccount(targetAccountNumber);
        if(!targetAccount) {
            std::cout << "Recipient account not found.\n";
            return;
        }

        std::cout << "Enter amount to transfer: ";
        std::cin >> amount;

        if(currentAccount->transfer(*targetAccount, amount)) {
            Transaction tx(Transaction::Type::TRANSFER,
                           currentAccount->getAccountNumber(),
                           amount, targetAccountNumber);
            tx.displayReceipt();
        } else {
            std::cout << "Transfer failed. Insufficient balance or invalid am
ount.\n";
        }
    }

    void ATM::processBalanceInquiry() {
        currentAccount->displayAccountInfo();
        Transaction tx(Transaction::Type::BALANCE_INQUIRY,
                       currentAccount->getAccountNumber(), 0);
        tx.displayReceipt();
    }
}
```

9. main.cpp (Application Entry Point)

```
#include "ATM.h"

int main() {
    KBZBank::ATM atm;
    atm.run();
    return 0;
}
```

✓ ရှင်းလင်းချက်

1. Namespace Structure
    - KBZBank namespace အတွင်းတွင် ATM system ၏ class များအားလုံးကို စုစည်းထားပါသည်
    - KBZ Bank အတွက်သီးသန့် implementation ဖြစ်ကြောင်း ဖော်ပြရန်

2. Class Design
    - Account: အကောင့်အချက်အလက်များနှင့် ငွေကြေးလုပ်ဆောင်ချက်များ
    - Transaction: ငွေလွှဲလုပ်ဆောင်ချက်မှတ်တမ်းနှင့် receipt ထုတ်ခြင်း
    - Database: အကောင့်များသိမ်းဆည်းထားသော database (နမူနာအဖြစ် memory တွင်သာ)
    - ATM: အဓိက application logic နှင့် user interface

3. Features
    - User authentication (account number + PIN)
    - ငွေထုတ်ခြင်း

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

- ငွေသွင်းခြင်း
- ငွေလွှဲခြင်း
- လက်ကျန်ငွေစစ်ဆေးခြင်း
- Transaction receipt ထုတ်ခြင်း

4. အသုံးပြုပုံ
    - Program ကို run သောအခါ user authentication လုပ်ရန် တောင်းဆိုမည်
    - Authentication အောင်မြင်ပါက main menu ပေါ်လာမည်
    - User ရွေးချယ်မှုအလိုက် လုပ်ဆောင်ချက်များကို ဆောင်ရွက်ပေးမည်
    - Sample account 3ခုဖြင့် အစပြုထားပါသည်

5. Security Notes
    - Real-world application တွင် PIN ကို encrypt လုပ်သိမ်းဆည်းရန်
    - Database ကို file system သို့မဟုတ် real database နှင့် အသုံးပြုရန်
    - Additional security measures များ ထပ်မံထည့်သွင်းရန်

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

❒ Console Chat Application

ဒီ application မှာ C++ သုံးပြီး console-based chat program တစ်ခုကို ဖန်တီးပါမယ်။ Client-server model ကို အသုံးပြုပြီး TCP socket programming နဲ့ ရေးသားထားပါတယ်။

⇨ Server Code (chat_server.cpp)

```
#include <iostream>
#include <vector>
#include <thread>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

using namespace std;

vector<SOCKET> clients;

void handleClient(SOCKET clientSocket) {
```

```
    char buffer[1024];
    while(true) {
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        if(bytesReceived <= 0) {
            cout << "Client disconnected" << endl;
            closesocket(clientSocket);
            return;
        }

        buffer[bytesReceived] = '\0';
        cout << "Message received: " << buffer << endl;

        // Broadcast to all clients
        for(SOCKET client : clients) {
            if(client != clientSocket) {
                send(client, buffer, bytesReceived, 0);
            }
        }
    }
}

int main() {
    cout << "Chat Server Starting..." << endl;

    // Initialize Winsock
    WSADATA wsData;
    if(WSAStartup(MAKEWORD(2, 2), &wsData) != 0) {
        cerr << "Winsock initialization failed" << endl;
        return 1;
    }

    // Create server socket
    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if(serverSocket == INVALID_SOCKET) {
        cerr << "Socket creation failed" << endl;
        WSACleanup();
        return 1;
    }

    // Bind socket
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
```

```
    serverAddr.sin_port = htons(54000);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    if(bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCK
ET_ERROR) {
        cerr << "Bind failed" << endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    // Listen for connections
    if(listen(serverSocket, SOMAXCONN) == SOCKET_ERROR) {
        cerr << "Listen failed" << endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    cout << "Server is running. Waiting for connections..." << endl;

    // Accept connections
    while(true) {
        SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);
        if(clientSocket == INVALID_SOCKET) {
            cerr << "Accept failed" << endl;
            continue;
        }

        clients.push_back(clientSocket);
        cout << "New client connected!" << endl;

        thread clientThread(handleClient, clientSocket);
        clientThread.detach();
    }

    closesocket(serverSocket);
    WSACleanup();
    return 0;
}
```

⇨ Client Code (chat_client.cpp)

```
#include <iostream>
#include <thread>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

using namespace std;

bool isRunning = true;

void receiveMessages(SOCKET sock) {
    char buffer[1024];
    while(isRunning) {
        int bytesReceived = recv(sock, buffer, sizeof(buffer), 0);
        if(bytesReceived <= 0) {
            cout << "Disconnected from server" << endl;
            isRunning = false;
            return;
        }

        buffer[bytesReceived] = '\0';
        cout << "Other: " << buffer << endl;
    }
}

int main() {
    cout << "Chat Client Starting..." << endl;

    // Initialize Winsock
    WSADATA wsData;
    if(WSAStartup(MAKEWORD(2, 2), &wsData) != 0) {
        cerr << "Winsock initialization failed" << endl;
        return 1;
    }
```

```
    // Create client socket
    SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
    if(sock == INVALID_SOCKET) {
        cerr << "Socket creation failed" << endl;
        WSACleanup();
        return 1;
    }

    // Connect to server
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(54000);
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if(connect(sock, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ER
ROR) {
        cerr << "Connection failed" << endl;
        closesocket(sock);
        WSACleanup();
        return 1;
    }

    cout << "Connected to server! Type your messages:" << endl;

    // Start receive thread
    thread receiveThread(receiveMessages, sock);
    receiveThread.detach();

    // Send messages
    string userInput;
    while(isRunning) {
        getline(cin, userInput);

        if(!isRunning) break;

        if(userInput == "exit") {
            isRunning = false;
            break;
        }

        send(sock, userInput.c_str(), userInput.size() + 1, 0);
```

```
    }

    closesocket(sock);
    WSACleanup();
    return 0;
}
```

✓ Program အလုပ်လုပ်ပုံ

1. Server Side
    - Server က port 54000 မှာ listen လုပ်ထားပါမယ်
    - Client တစ်ယောက်ချင်းစီကို thread တစ်ခုစီနဲ့ ကိုင်တွယ်ပါမယ်
    - Client ကပို့လိုက်တဲ့ message ကို အခြား client တွေဆီကို broadcast လုပ်ပေးပါမယ်
2. Client Side
    - Server ဆီကို connect လုပ်ပြီး message တွေပို့နိုင်ပါမယ်
    - Message တွေကို receive လုပ်ဖို့အတွက် အထူး thread တစ်ခုသုံးထားပါမယ်
    - "exit" လို့ရိုက်ရင် program ကို ထွက်ပါမယ်
3. Communication
    - Client တစ်ယောက်က message ပို့ရင် server က အခြား client တွေဆီကို ပြန်ပို့ပေးပါမယ်
    - TCP socket ကို အသုံးပြုထားတာကြောင့် reliable communication ရပါမယ်

✓ Compile နည်း

Windows မှာ compile လုပ်ဖို့:

1. Server အတွက်:

```
g++ chat_server.cpp -o server -lws2_32
```


မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

2. Client အတွက်:

```
g++ chat_client.cpp -o client -lws2_32
```

✓ အသုံးပြုနည်း

1. Server ကို အရင် run ပါ:

```
.\server.exe
```

2. Client တွေကို သီးသန့် console တွေမှာ run ပါ:

```
.\client.exe
```

3. Client တစ်ခုကနေ message ရိုက်ထည့်ရင် အခြား client တွေဆီကို ရောက်သွားပါမယ်

✓ ထပ်မံဖြည့်စွက်နိုင်သော Features များ

1. Username Support
   - User တွေကို name တောင်းပြီး message တွေမှာ username ပါအောင် လုပ်နိုင်ပါတယ်
2. Private Messaging
   - Specific user တစ်ယောက်ဆီကို message ပို့နိုင်အောင် လုပ်နိုင်ပါတယ်
3. Room/Chat Groups
   - အခြား chat room တွေကို ဖန်တီးပြီး join လုပ်နိုင်အောင် လုပ်နိုင်ပါတယ်
4. Encryption
   - Message တွေကို encrypt လုပ်ပြီး ပို့နိုင်အောင် လုပ်နိုင်ပါတယ်

မှတ်ချက်: ဒီ program က Windows အတွက်ရေးထားပါတယ်။ Linux/macOS မှာ အသုံးပြုချင်ရင် Winsock အစား sys/socket.h ကို အသုံးပြုရပါမယ်။

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

ကျေးဇူးတင်စကား

စာဖတ်သူက ဒီနေ့ ဒီစာမှက်နှာအထိ အမှန်တကယ် အချိန်ပေးပြီး အဆင့်ဆင့်လေ့လာခဲ့တယ်ဆိုရင် ကျနော် စာရေးသူကနေ အများကြီးကျေးဇူးတင်တယ်လို့ ပြောပါရစေ။ အိုင်တီနယ်ပယ်ကို ဝင်လာတဲ့ သူ အယောက် ၁၀၀ မှာ ၁၀ ယောက်က productive programmer ဖြစ်ဖို့မလွယ်လှပါဘူး။ လေး ငါးယောက်လောက်ပဲ ဒီလောက်ကမှာ ဆက်လက်ရှင်သန်နိုင်ကြလို့ပါ။ ဒီနယ်ပယ်က လူတိုင်းအတွက် မဟုတ်ပါဘူး။ အိုင်တီကို ဇောက်ချလုပ်တော့မယ်လို့ စိတ်ဆုံးဖြတ်ထားတဲ့သူ၊ ဇွဲရှိတဲ့သူ ၊ ပညာရေး background ကောင်းတဲ့သူ၊ အိုင်တီအတွက် ပံပိုးမှုကောင်းတဲ့သူ၊ အကုန်အကျခံနိုင်တဲ့သူ တွေသာလျှင် programmer ဖြစ်လာကြတာပါ။ အနုပညာရှူထောင့်ကနေ ပြောရမယ်ဆိုရင် အိုင်တီက panio လောကနဲ့ ဆင်ပါတယ်။ အနည်းဆုံး ၁၀ နှစ်လောက် အချိန်မပေးပဲနဲ့ စင်ပေါ်မရောက်ပါဘူး။ အိုင်တီလောက မှာလဲ အနည်းဆုံး လေး ငါးနှစ်လောက် အချိန်မပေးပဲ ဘယ်မှ မရောက်ပါဘူး။ အဲဒါကြောင့် ဒီနေ့အထိ ဒီစာအုပ်ထဲက လေ့ကျင့်ခန်းတွေ ကုန်တဲ့အထိ စာဖတ်သူ အချိန်ပေး၊ အပင်ပန်းခံလိုက်တဲ့ လေ့လာမှုတွေအတွက် အရမ်းကို ဂုဏ်ယူပါတယ်။

အိုင်တီနယ်ပယ်ထဲက တစ်ချို့ဆရာတွေ coding ကို အရေးမကြီးသလိုလို ပေါ့ပေါ့ပါးပါးနဲ့ ဖြတ်သန်းလို့ရမယ့် career လိုလို ပြောကြဆိုကြတာတွေ ခဏခဏ ကြုံဖူးပါတယ်။ coding ကို အလေးမပေးတဲ့ အိုင်တီနယ်ပယ်ထဲက မိတ်ဆွေ သူငယ်ချင်းတွေကို ခကြာခဏ တွေ့ဖူးပါတယ်။ ကျနော့်ရဲ့ ၁၅ နှစ်တာ အိုင်တီဖြတ်သန်းမှုကို ကြည့်ပြီး ပြောရမယ်ဆိုရင် အဲဒီ coding မရလဲ ဘာမှ မဖြစ်ဘူး။ coding မရလဲ အဆင်ပြေပါတယ် ဆိုတဲ့ အိုင်တီနယ်ပယ်ထဲမှာ နေနေတဲ့ သူတွေသည် သူတို့ အားနည်းချက်တွေကို ဖုံးကွယ်လိုတဲ့ အတွက် ပြောခြင်းသာ ဖြစ်ပါတယ်။ Coding က အိုင်တီးလောက မှာ အရေးအကြီးဆုံးအောင်မြင်မှု key ဖြစ်ပါတယ်။ အိုင်တီ career ရဲ့ အသက်သွေးဖြစ်ပါတယ်။ ဒီလို ခက်ခက်ခဲခဲ လုပ်ရတဲ့ ပုဒ်စာတွေ၊ စိန်ခေါ်မှုတွေနဲ့ ပြည့်နေတဲ့ လေ့ကျင့်ခန်းတွေကို ပြီးမြောက်အောင် လုပ်ခဲ့တဲ့ အတွက် ထပ်ပြီး congratualtions ပါလို့ ပြောချင်ပါတယ်။

■ References

1. "C++ Primer" – Stanley B. Lippman, Josée Lajoie, Barbara E. Moo
2. "Programming: Principles and Practice Using C++" – Bjarne Stroustrup
3. "Starting Out with C++ from Control Structures to Objects" – Tony Gaddis
4. "C++ for Beginners" – Herbert Schildt
5. "Effective C++" – Scott Meyers
6. "Effective Modern C++" – Scott Meyers
7. "C++ Templates: The Complete Guide" – David Vandevoorde, Nicolai M. Josuttis
8. "The C++ Standard Library: A Tutorial and Reference" – Nicolai M. Josuttis
9. https://cplusplus.com/doc/tutorial/
10. https://www.tutorialspoint.com/cplusplus
11. https://www.w3schools.com/cpp/
12. https://openai.com/index/chatgpt
13. https://chat.deepseek.com

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။

■ စားရေးသူ၏ ကိုယ်ရေးအကျဉ်း

ဤစာအုပ်ကိုရေးသားပြုစုသူကတော့ ကျွန်တော် တင်မိုင်ဇော် ဖြစ်ပါတယ်။ စာရေးသူက မြစ်ကြီးနားမြို့ အထက (၁) မှာ အထက်တန်းကို ၁၉၉၄ မှာ အောင်မြင်ခဲ့ပါတယ်။ ပြီးတော့ ကျွန်တော် ဖိလိပိုင်နိုင်ငံ University of the Philippines (Los Banos) မှာ B.Sc Computer Science ကိုဆက်လက်ပညာသင်ယူခဲ့ပါတယ်။ ၂၀၀၄ မှာ B.Sc Computer Science နဲ့ ဘွဲ့ရကျောင်းပြီး ခဲ့ပါတယ်။ ၂၀၀၅ မှာ မြန်မာပြည်ပြန်လာပြီး မြစ်ကြီးနား ဇာတိမြို့မှာ Northern City Computer Training Center ကွန်ပျူတာသင်တန်းကျောင်းကို စတင်တည်ထောင်ဖွင့် လှစ်ခဲ့ပါတယ်။ လေ့လာဆည်းပူးခဲ့တဲ့ အိုင်တီ programming ဘာသာရပ်တွေကို သင်ကြားပို့ချခဲ့တာ ဖြစ်ပါတယ်။ ၂၀၀၉ မှာ မလေးရှားနိုင်ငံ Kualalumpur မှာ Zepto IT Solution လို့ခေါ်တဲ့ အိုင်တီ Company မှာ Software Developer (programmer) အဖြစ် ၃ နှစ်တာ အလုပ်လုပ်ခဲ့ပါတယ်။ ၂၀၁၂မှာ ကျန်းမာရေး အခြေနေ ကြောင့် ရန်ကုန်မြို့ကို ပြန်လာပြီး ဆေးကုသရင်း ရန်ကုန်မြို့မှာပဲ ValueStar Computer Institute မှာ ၆ နှစ်တာ IT Lecturer အနေနဲ့ရော IT Department Head အနေနဲ့ရော ဆက်လက်ခြေချခဲ့ပါတယ်။ ၂၀၁၇ မှာ ကိုယ်ပိုင် အိုင်တီသင်တန်းကျောင်းအဖြစ် Northern City Center နာမည်နဲ့ အရင်က မြစ်ကြီးနားမှာ တည်ထောင်ခဲ့ဖူး တဲ့ နာမည်ကို ပြန်လည်အသုံးပြုကာ ဖွင့်လှစ်ထားပြီး ၂၀၂၅ လက်ရှိအချိန်အထိ သင်တန်းကျောင်းကို ဦးစီးလုပ်ကိုင်နေဆဲ ဖြစ်ပါတယ်။

■ စာရေးသူ၏ အိုင်တီအတွေ့အကြုံ မှတ်တမ်းများ

✓ Popular Web projects –

- Japan Used Car Sales & Show room
  https://www.sbtjapan.com/sbt-myanmar/
- Myanmar Traditional Boxing
  https://www.myanmartraditionalboxing.com.mm
- Myanmar Agriculture Machinery & Products
  https://tptyeeshinn.com.mm
- Real Estate
  https://www.saikhungnoung.com
- Malaysia Minimart
  https://familystore.com.my
- China Products & Fashion Sales
  https://youhome.space/
- Online Payment System
  https://ucapay.com.mm

✓ Popular Point of Sales Application Projects –

- Malaysia Family Store Desktop Application and Mobile Application
- Myitkyina iGu Café & Gallery Desktop Application
- Myitkyina Hospital Blood Bank Desktop Application
- Myitkyina Fuji Food & Drinks Desktop Application
- Yangon Joyzone Stock Controller Desktop Application and Mobile Application
- Yangon Malihka Restaurant Desktop Application and Mobile Application
- Yangon Yuri Café Mobile Application and Mobile Application
- Yangon Gracevines Agarwood Desktop Application and Mobile Application

■ စာရေးသူ၏ Documentary Photos

မြန်မာပြည်သားတိုင်း ကွန်ပျူတာ တတ်ရမည်။